Modular Models of Monoids with Operations

ZHIXUAN YANG, Imperial College London, United Kingdom
NICOLAS WU, Imperial College London, United Kingdom

Inspired by algebraic effects and the principle of *notions of computations as monoids*, we study a categorical framework for *equational theories* and *models* of monoids equipped with operations. The framework covers not only algebraic operations but also scoped and variable-binding operations. Appealingly, in this framework both theories and models can be modularly composed. Technically, a general monoid-theory correspondence is shown, saying that the category of *theories of algebraic operations* is equivalent to the category of monoids. Moreover, more complex forms of operations can be coreflected into algebraic operations, in a way that preserves initial algebras. On models, we introduce *modular models* of a theory, which can interpret abstract syntax in the presence of other operations. We show constructions of modular models (i) from monoid transformers, (ii) from free algebras, (iii) by composition, and (iv) in symmetric monoidal categories.

Additional Key Words and Phrases: equational systems, Σ-monoids, effects, modularity, monad transformers

1 INTRODUCTION

In his seminal work, Moggi [1989a,b, 1991] pioneered modelling *notions of computation*, which are now more commonly referred to as *computational effects*, using *monads* and *monad transformers*. The understanding of this approach is later deepened by Plotkin and Power [2001, 2002, 2004], who characterise many monads that model computational effects as arising from *equational theories* of some primitive effectful operations and equational laws charactering their interaction.

These ideas from denotational semantics are quickly adopted by the functional programming community as an abstraction for effectful programming [Filinski 1994; Wadler 1995]. Many refinements have been proposed for making programming with effects more *modular*. A fundamental idea is to *separate syntax from semantics*, so syntactic effectful programs can be written without resorting to a specific semantics, and (possibly many kinds of) semantics can be given later. Building on this, there are further two kinds of modularity that are both desirable but are sometimes conflated:

Syntactic modularity is where syntactic specifications of smaller languages can be composed to form syntactic specifications of larger languages. For example, in some frameworks the syntax for mutable state and the syntax for exceptions may be combined together.

Semantic modularity is where semantic models of a language can be reused, combined, or extended when new syntax is introduced into the language. For example, in some frameworks a model of state can be reused when new operations are added to the language.

In terms of this classification, variants of free monads provide syntactic modularity [Kiselyov and Ishii 2015; Swierstra 2008; Voigtländer 2008]. Type classes of monads with operations [Liang et al. 1995] implemented in Haskell [Gill and Kmett 2012] also achieve syntactic modularity.

On the other hand, *monad transformers* [Moggi 1989a], *layered monads* [Filinski 1999], combining monads by coproducts [Ghani and Uustalu 2004], and the work by Jaskelioff and Moggi [2010] on lifting operations through monad transformers are about semantic modularity.

Effect Handlers. Notably, *effect handlers*, introduced by Plotkin and Pretnar [2013] and further developed by many people, achieve both kinds of modularity, which explains their quick adoption in many programming languages. But as analysed in [Wu et al. 2014; Yang et al. 2022], only *algebraic operations* [Plotkin and Power 2003] enjoy modularity in this approach. Here an algebraic operation

Authors' addresses: Zhixuan Yang, s.yang20@imperial.ac.uk, Department of Computing, Imperial College London, United Kingdom; Nicolas Wu, n.wu@imperial.ac.uk, Department of Computing, Imperial College London, United Kingdom.

on a monad $M : \mathscr{C} \to \mathscr{C}$ is a natural transformation $\alpha : A \circ M \to M$ for an endofunctor $A : \mathscr{C} \to \mathscr{C}$, typically a polynomial functor $P \times (-)^O$, such that it is compatible with monad multiplication:

$$\mu^M \cdot (\alpha \circ M) \;=\; \alpha \cdot (A \circ \mu^M) \;:\; A \circ M \circ M \to M.$$

Non-algebraic operations, such as exception catching or parallel composition, *can* be programmed as handlers, but they do not have the benefit of (syntactic or semantic) modularity.

A line of research seeks to lift this restriction on effect handlers [Bach Poulsen and van der Rest 2023; Piróg et al. 2018; Wu et al. 2014; Yang et al. 2022] by considering signatures/theories of broader ranges of operations and their models. For example, Piróg et al. [2018] consider operations on monads M of the following form for some $A : \mathscr{C} \to \mathscr{C}$, which they call *scoped operations*,

$$s : \int^{X \in \mathscr{C}} A(MX) \times (M-)^X \cong A \circ M \circ M \to M. \tag{1}$$

Typically, the functor A is $(-)^n$ for some natural number n. The intuition is that s is an operation delimiting n scopes: $A(MX)$ is the computation inside the scopes, returning some existentially quantified type X; $(M-)^X$ is the computation after these scopes. For example, the operation of *exception catching* delimits two scopes, one for 'try' and the other for 'catch', so it can be modelled with $A = (-)^2 \cong (- \times -)$. The coend in (1) is isomorphic to $A \circ M \circ M$ by the co-Yoneda lemma.

However, existing work in this direction only considers syntactic modularity (with the exception [Wu et al. 2014]). The root of the difficulty with semantic modularity is that only algebraic operations have a canonical lifting along a monad morphism: given a monad morphism $f : M \to N$ and an algebraic operation $\alpha : A \circ M \to M$ on M, there is a unique algebraic operation $\overline{\alpha} : A \circ N \to N$:

$$\overline{\alpha} = (\;A \circ N \xrightarrow{A \circ \eta^M \circ N} A \circ M \circ N \xrightarrow{\alpha \circ N} M \circ N \xrightarrow{f \circ N} N \circ N \xrightarrow{\mu^M} N\;) \tag{2}$$

such that $f : M \to N$ is an algebra homomorphism from α to $\overline{\alpha}$. In contrast, if the operation takes the form (1) or more generally the form $\alpha : \Sigma M \to M$ for a functor $\Sigma : \mathbf{Endo}(\mathscr{C}) \to \mathbf{Endo}(\mathscr{C})$, we do not have a formula to define a meaningful $\overline{\alpha} : \Sigma N \to N$ from $f : M \to N$ and $\alpha : \Sigma M \to M$.

Overview. The present paper has two aims/parts: the first one is to develop a unifying account of equational theories of (algebraic and non-algebraic) effectful operations; the second one is to develop a notion of *modular models* for equational theories of operations as well as constructions of modular models. These two parts together achieve both syntactic and semantic modularity in the style of effect handlers, but for a wider range of operations.

The main definition of the first part (§2–4) is *monoidal theory families*, which are categories of equational theories of *monoids with operations*, such that the family is closed under coproducts, and every theory in the family admits free algebras. Examples include the family of *algebraic operations*, and the family of *scoped operations*, and the family of *variable-binding operations*.

The main definition of the second part (§5–6) is *modular models* M of some theory $\tilde{\Gamma}$ in a theory family $\mathcal{F}$. Every M is basically a family of functors $\tilde{\Sigma}$-$\mathbf{Alg} \to (\tilde{\Sigma} + \tilde{\Gamma})$-$\mathbf{Alg}$ (oplax-) natural in $\tilde{\Sigma} \in \mathcal{F}$. In words, a modular model *sends monoids with operations to monoids with more operations.*

Example. We sketch a small example here to demonstrate the ideas. Let $\mathscr{E}$ be the monoidal category $\langle \mathbf{Endo}_f(\mathbf{Set}), \circ, \mathrm{Id}\rangle$ of finitary endofunctors on sets. The equational theory Ec of monads M with *exception catching* has two additional operations besides those of monads:

$$\mathit{throw} : 1 \to M \qquad \text{and} \qquad \mathit{catch} : (\mathrm{Id} \times \mathrm{Id}) \circ M \circ M \cong (M \times M) \circ M \to M,$$

and for now let us ignore equations on these operations. The operation *throw* is for throwing an exception. The operation *catch* is for catching an exception in a scope and handling it, which is an instance of (1) with $A = \mathrm{Id} \times \mathrm{Id}$. As we briefly mentioned earlier, the product $M \times M$ in the signature of *catch* is a pair of computations, one for the program p whose exceptions are caught, the other one for the program h handling the exception; the $\circ M$ after $M \times M$ in the signature of *catch* is an *explicit continuation* argument k. Thus *catch* $(\langle p, h\rangle, k)$ is understood as handling the exception in p using h, and then continues as k. The explicit continuation is exactly the trick to workaround the limitation of algebraic operations (we will say more about it in §4).

All theories of *monads with some scoped operations* are collected as a category $\mathbf{SCP}(\mathscr{E})$, which we call the *monoidal theory family of scoped operations* over $\mathscr{E}$, whose arrows are *translations* of theories. The category $\mathbf{SCP}(\mathscr{E})$ has finite coproducts by taking the coproduct of the signatures and equations. Moreover, each theory in $\mathbf{SCP}(\mathscr{E})$ has free algebras, in particular initial algebras. For example, the initial algebra of Ec is the initial one among all *monads with throw and catch*. The carrier of the initial algebra $\mu\mathrm{Ec} : \mathbf{Set} \to \mathbf{Set}$ can be characterised as the initial solution to

$$X \;\cong\; \mathrm{Id} + 1 + (X \times X) \circ X \;\in\; \mathbf{Endo}_f(\mathbf{Set}).$$

The monad $\mu\mathrm{Ec}$ models *syntactic programs* with exception throwing and catching.

A (strict) *modular model* of the theory Ec is a family of functors $M_{\tilde{\Sigma}} : \tilde{\Sigma}\text{-}\mathbf{Alg} \to (\tilde{\Sigma} + \mathrm{Ec})\text{-}\mathbf{Alg}$, natural in $\tilde{\Sigma} \in \mathbf{SCP}(\mathscr{E})$. Here $(-)\text{-}\mathbf{Alg}$ is the functor $\mathbf{SCP}(\mathscr{E}) \to \mathbf{CAT}$ sending each theory to the category of its models. One possible modular model of Ec is to send every $\tilde{\Sigma}$-algebra carried by A to $A \circ (1 + \mathrm{Id})$ equipped with operations in $\tilde{\Sigma}$ and Ec. In the same way that there may be many handlers of the same algebraic operation, we also have choices over how *catch* and *throw* act on $A \circ (1 + \mathrm{Id})$. Besides the 'standard' semantics (detailed in Example 6.3), we may also have non-standard semantics such as re-trying the program after the handling program is executed, or the semantics that the exceptions thrown by the handling program are recursively handled.

The benefit of a modular model M of Ec over an ordinary model of Ec is that M allows us to interpret syntactic programs $\mu(\tilde{\Sigma} + \mathrm{Ec})$ involving throwing and catching mixed with *any other operations* $\tilde{\Sigma}$ in $\mathbf{SCP}(\mathscr{E})$. By the initiality of $\mu(\tilde{\Sigma} + \mathrm{Ec})$, there is a morphism:

$$h : \mu(\tilde{\Sigma} + \mathrm{Ec}) \to M_{\tilde{\Sigma}}(\mu\tilde{\Sigma}) \quad \text{in} \quad (\tilde{\Sigma} + \mathrm{Ec})\text{-}\mathbf{Alg},$$

which interprets Ec but leaves $\tilde{\Sigma}$-operations uninterpreted. In this way, we achieve syntactic and semantic modularity in the style of effect handlers, but for the non-algebraic operation *catch*.

Paper Organisation. In §2, we review monoidal categories and a metalanguages for them. In §3, we recap Fiore and Hur [2009]'s *equational systems* and define *functorial translations* between them, making them a category, and show some constructions of colimits in this category. In §4, we define *monoidal theory families* and study the connections between some notable examples. In §5, we introduce *modular models* of a theory and show how they can be used for interpreting syntax. In §6, we show general constructions and examples of modular models.

Along these lines, we make the following contributions:

(1) We show in §3 a syntactic way to present Fiore and Hur [2009]'s *equational systems*, based on a *metalanguage for monoidal categories* by Jaskelioff and Moggi [2010].

(2) We introduce in §3.4 *functorial translations* between equational systems, and we show that a subcategory of equational systems is cocomplete, allowing equational systems to be modularly composed, and every functorial translation has a left adjoint, producing 'relative free algebras'.

(3) We show a general *monoid-theory correspondence* (Theorem 4.3)—under certain conditions on the monoidal category $\mathscr{E}$, the theory family $\mathbf{ALG}(\mathscr{E})$ of monoids with algebraic operations is equivalent to the category $\mathbf{Mon}(\mathscr{E})$ of monoids in $\mathscr{E}$, generalising the classical correspondence between finitary monads and presentations of finitary algebraic theories.

(4) We show that the theory family $\mathbf{ALG}(\mathscr{E})$ is a coreflective subcategory of the family of all theories with cocontinuous signatures and contexts, and the coreflection *preserves initial algebras* (Theorem 4.4). This means that algebraic operations are enough for modelling abstract syntax.

(5) We show several general constructions and examples of modular models: modular models of algebraic or scoped operations can be obtained from monoid transformers (§6.1) based on results by Jaskelioff and Moggi [2010]; *free modular models* can be obtained from relative free algebras; modular models can be *composed*; modular models in symmetric monoidal categories can be obtained from ordinary models in two different ways, dependent and independent combinations (§6.4); *phased computation* can be taken as a modular model.

2 MONOIDS, MONOIDAL CATEGORIES, AND A METALANGUAGE

To put our work in context, we first review the concept of *monoids in monoidal categories* and some examples that are relevant in programming language theory (§2.1). We then introduce a *metalanguage for monoidal categories*, adapted from Jaskelioff and Moggi [2010], which we will use in later sections for describing constructions in monoidal categories conveniently (§2.2).

2.1 Monoidal Categories and Monoids

A *monoidal category* is a category $\mathscr{E}$ equipped with a functor $\square : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$, called the *monoidal product*, an object $I \in \mathscr{E}$, called the *monoidal unit*, and three natural isomorphisms

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

satisfying some coherence axioms [Mac Lane 1998, §VII.1]. A monoidal category is *(right) closed* if all functors $- \square A$ have right adjoints $-/A : \mathscr{E} \to \mathscr{E}$.

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

$$\begin{tikzcd}
(M \square M) \square M \arrow[rr, "\alpha_{M,M,M}"] \arrow[d, "\mu \square M"'] & & M \square (M \square M) \arrow[d, "M \square \mu"] \\
M \square M \arrow[r, "\mu"'] & M & M \square M \arrow[l, "\mu"]
\end{tikzcd}
\qquad
\begin{tikzcd}
I \square M \arrow[d, "\eta \square M"'] & M \arrow[l, "\lambda_M"'] \arrow[r, "\rho_M"] \arrow[d, equal] & M \square I \arrow[d, "M \square \eta"] \\
M \square M \arrow[r, "\mu"'] & M & M \square M \arrow[l, "\mu"]
\end{tikzcd} \tag{3}$$

Below we review several monoidal categories which are relevant in programming, especially several monoidal structures on endofunctors, in which monoids model different flavours of *notions of computations*, and they will be the main applications of the theory developed in this paper.

Monads. The category $\mathbf{Endo}(\mathscr{C})$ 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 1989b, 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.

However, the category $\mathbf{Endo}(\mathscr{C})$ 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 $\mathbf{Set}$. In particular, $\mathbf{Endo}(\mathbf{Set})$ is not closed with respect to either cartesian products or functor composition; and some objects in **Endo**(**Set**), such as the *continuation monad* $R^{(R^-)}$, do not have free monoids over them. These will be problematic when considering algebraic theories on **Endo**($\mathscr{C}$). Fortunately, these problems can be rectified by restricting to the subcategory of *finitary endofunctors*.

Finitary Monads. An endofunctor $F \in \mathbf{Endo}(\mathbf{Set})$ is called *finitary* if it preserves *filtered colimits*. A useful characterisation of finitariness is that *we lose no information if we restrict F to finite sets*. Precisely, F is finitary exactly when we first restrict F as $F \circ V : \mathbf{Fin} \longrightarrow \mathbf{Set}$ on the full subcategory of finite sets, where $V : \mathbf{Fin} \longrightarrow \mathbf{Set}$ 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 $\mathbf{Endo}_f(\mathbf{Set}) \subseteq \mathbf{Endo}(\mathbf{Set})$ denotes the full subcategory of finitary endofunctors:

$$\begin{tikzcd} \mathbf{Endo}_f(\mathbf{Set}) \arrow[r, shift right, "-\circ V"'] \arrow[r, phantom, "\cong"] & \mathbf{Set}^{\mathbf{Fin}} \arrow[l, shift right, "\mathbf{Lan}_V -"'] \end{tikzcd} \tag{4}$$

The category $\mathbf{Endo}_f(\mathbf{Set})$ inherits the monoidal structure $\langle \circ, \mathrm{Id} \rangle$ of **Endo**(**Set**). What is new is that the monoidal structure is closed, and functor composition in $\mathbf{Endo}_f(\mathbf{Set})$ is itself a finitary functor. These conditions suffice to guarantee that every object in $\mathbf{Endo}_f(\mathbf{Set})$ have free monoids.

Monoids in $\mathbf{Endo}_f(\mathbf{Set})$ are called *finitary monads* on **Set**, and they are known to be equivalent to (finitary) *Lawvere theories* [Lawvere 1963; Linton 1966] and *abstract clones* [Cohn 1981].

Apart from modelling computational effects, a related but slightly different application of monoids in $\mathbf{Endo}_f(\mathbf{Set})$, or equivalently $\mathbf{Set}^{\mathbf{Fin}}$, is modelling *abstract syntax with variable binding* [Fiore and Szamozvancev 2022; Fiore et al. 1999]. In this case, a monoid $M \in \mathbf{Set}^{\mathbf{Fin}}$ is understood as a *variable set* of terms indexed by the number of *variables* in the context. Under the equivalence (4), the monoidal structure $\langle \circ, \mathrm{Id} \rangle$ of $\mathbf{Endo}_f(\mathbf{Set})$ is equivalent to $\langle \bullet, V \rangle$ on $\mathbf{Set}^{\mathbf{Fin}}$ where

$$Vn = n \qquad\qquad \text{and} \qquad\qquad (F \bullet G)n = \int^{m \in \mathbf{Fin}} (Fm) \times (Gn)^m. \tag{5}$$

The monoid unit $V \longrightarrow M$ is then embedding *variables* as M-terms, and the monoid multiplication $M \bullet M \longrightarrow M$ is *simultaneous substitution* of terms for variables. Moreover, the right adjoint $-/G$ to $- \bullet G$ is given by the end formula: $(F/G)n = \int_{m \in \mathbf{Fin}} \prod_{\mathbf{Set}(n, Gm)} Fm$.

More generally, we can replace **Set** in the situation $\mathbf{Endo}_f(\mathbf{Set}) \cong \mathbf{Set}^{\mathbf{Fin}}$ above with any *locally κ-presentable* (lκp) category $\mathscr{C}$ for a regular cardinal κ, **Fin** with the subcategory $\mathscr{C}_\kappa$ of *κ-presentable objects* in $\mathscr{C}$, and finitary functors with *κ-accessible* functors $\mathbf{Endo}_\kappa(\mathscr{C})$ [Adamek and Rosicky 1994]. This results in a cocomplete closed monoidal category $\langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$, on which $\circ$ is a κ-accessible functor. Examples of lκp categories include all presheaf categories $\mathbf{Set}^{\mathscr{D}}$ for $\kappa = \aleph_0$ (they are called locally *finitely* presentable when $\kappa = \aleph_0$), and the category $\omega\mathbf{Cpo}$ of ω-complete partial orders for $\kappa = \aleph_1$. Moreover, when $\mathscr{C}$ is lκp, it is automatically lλp for any $\lambda > \kappa$.

Locally κ-presentable categories provide a nice setting for doing algebraic theories and are general enough for various applications in programming languages. We will rely on some standard results about lκp categories to guarantee the existence of free algebras, but the readers do not need to know any technical details about them to understand most of the present paper.

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

The category $\mathbf{Endo}_\kappa(\mathscr{C})$ is also cartesian closed whenever $\mathscr{C}$ is lκp and cartesian closed. The cartesian unit and product in $\mathbf{Endo}_\kappa(\mathscr{C})$ are defined pointwise: $1n = 1_{\mathscr{C}}$ and $(F \times G)n = Fn \times Gn$. A computational interpretation of cartesian monoids in $\mathbf{Endo}_\kappa(\mathscr{C})$ is that they model *notions of* *independent computations*: the cartesian product $M \times M \to M$ 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.

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

$$(F \star G)n = \int^{m,k \in \mathbf{Set}_\kappa \times \mathbf{Set}_\kappa} Fm \times Gk \times n^{m \times k}. \tag{6}$$

Informally, the Day convolution $F \star G$ 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 $- \star G$ given by $G \mathrel{-\!\!\star} F = \int_{n \in \mathbf{Set}_\kappa} (F(- \times n))^{Gn}$.

Monoids for $\langle \star, \mathrm{Id} \rangle$ are called *applicative functors* or simply *applicatives* [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.

Graded Monads. A 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 2022]. For example, the grades can be a set of operations that a computation may invoke, or it can be the number of nondeterministic choices that a computation makes. Precisely, let $\langle \mathbb{G}, \cdot, 1 \rangle$ be any small strict monoidal category, whose objects are called grades. A *finitary* $\mathbb{G}$*-graded monad* [Kura 2020] is a functor $M : \mathbb{G} \to \mathbf{Endo}_f(\mathbf{Set})$ equipped with

$$\eta : \mathrm{Id} \to M1 \qquad\qquad \mu_{a,b} : (Ma) \circ (Mb) \to M(a \cdot b) \quad \text{for all } a, b \in \mathbb{G} \tag{7}$$

satisfying laws similar to those of monads. For example, for tracking operations performed by a computation, $\mathbb{G}$ 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, $\mathbb{G}$ can be the poset $\langle \mathbb{N}, \leqslant \rangle$ with monoidal structure $\langle 0, + \rangle$.

Finitary graded monads are equivalent to monoids in the functor category $\mathbf{Endo}_f(\mathbf{Set})^{\mathbb{G}}$ equipped with the following variation of the Day tensor product:

$$I = \coprod_{\mathbb{G}(1,-)} \mathrm{Id} \qquad\qquad F \star G = \int^{a,b \in \mathbb{G}} \coprod_{\mathbb{G}(a \cdot b, -)} (Fa \circ Gb). \tag{8}$$

A proof of the equivalence can be found in the supplementary material.

2.2 A Metalanguage for Monoidal Categories

When the monoidal category gets complex, commutative diagrams like those in (3) can be unwieldy, we will use a typed calculus introduced by Jaskelioff and Moggi [2010] as a *metalanguage* to denote constructions in monoidal categories, in the same way that λ-calculus can be used to denote constructions in cartesian closed categories. The use of the calculus not only provides a convenient syntax, but also provides useful intuition as if we were working in the category of sets.

Types and Terms. Assume a set Ba of base types ranged over by α. The types A, B of the metalanguage are inductive generated by the rule $A, B ::= \alpha \mid A \mathbin{\square} B \mid \mathrm{I}$. Further assume a set Pr of primitive operations ranged over by f, each associated with two types $f : A \to B$. The well typed terms of the calculus are generated by the typing rules in Figure 1 (there is no need to consider raw terms for our purposes). The type system is a substructural one, resembling Polakow and Pfenning [1999]'s *intuitionistic non-commutative linear logic*, since the language is to be interpreted

$$\frac{}{x : A \vdash x : A} \qquad \frac{f : A \longrightarrow B \in \mathsf{Pr} \qquad \Gamma \vdash t : A}{\Gamma \vdash f(t) : B} \qquad \frac{}{\vdash * : \mathsf{I}} \qquad \frac{\Gamma_1 \vdash t_1 : A \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1, \Gamma_2 \vdash (t_1, t_2) : A \Box B}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{I} \qquad \Gamma_l, \Gamma_r \vdash t_2 : A}{\Gamma_l, \Gamma, \Gamma_r \vdash \mathsf{let}\ * = t_1\ \mathsf{in}\ t_2 : A} \qquad \frac{\Gamma \vdash t_1 : A_1 \Box A_2 \qquad \Gamma_l, x_1 : A_1, x_2 : A_2, \Gamma_r \vdash t_2 : B}{\Gamma_l, \Gamma, \Gamma_r \vdash \mathsf{let}\ (x_1, x_2) = t_1\ \mathsf{in}\ t_2 : B}$$

Fig. 1. Well typed terms for the metalanguage

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

Fig. 2. Denotational semantics for the metalanguage. The underscores stand for appropriate canonical isomorphisms built from associators α and unitors λ, ρ to make the domain and codomain match.

in monoidal categories rather than only cartesian categories. The rules in Figure 1 always have ambient contexts Γ, Γ_l and Γ_r whenever possible so that a cut rule is admissible:

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

Denotation. Given a monoidal category $\mathscr{E}$, if an interpretation $[\![\alpha]\!] \in \mathbf{Ob}(\mathscr{E})$ is assigned to each base type $\alpha \in \mathsf{Ba}$, all types and typing contexts can be interpreted as objects in $\mathscr{E}$:

$$[\![\mathsf{I}]\!] = I, \qquad [\![A \Box B]\!] = [\![A]\!] \Box [\![B]\!], \qquad [\![\cdot]\!] = I, \qquad [\![\Gamma, x : A]\!] = [\![\Gamma]\!] \Box [\![A]\!].$$

Given an interpretation $[\![f]\!] : [\![A]\!] \longrightarrow [\![B]\!]$ in $\mathscr{E}$ for each primitive operation $f : A \longrightarrow B$, all well typed terms $\Gamma \vdash t : A$ can be interpreted as morphisms $[\![t]\!] : [\![\Gamma]\!] \longrightarrow [\![A]\!]$ inductively. The interpretation for each typing rule is given in Figure 2. When we want to be explicit about the denotation of base types and primitive operations that define $[\![-]\!]$, we write them in the subscript like $[\![-]\!]_{\{\alpha \mapsto A\}}$.

Example 2.1. Assume a base type M and a primitive operation $\mu : M \Box M \longrightarrow M$. The first diagram in the laws of monoids (3) can be denoted by the following pair of terms of the same type (throughout this paper, we write $\Gamma \vdash t_1 = t_2 : A$ for two terms t_1 and t_2 encoding an equation):

$$\frac{\mu : M \Box M \longrightarrow M}{x : M, y : M, z : M \vdash \mu(\mu(x, y), z) \ = \ \mu(x, \mu(y, z)) : M} \tag{9}$$

which looks the same as the usual associativity law for a binary operation μ in **Set**, but the metalanguage can be interpreted in all monoidal categories.

Extensions. Sometimes we work in monoidal categories with additional structure, and in this case we extend the calculus with new syntax for the additional structure. For example, when we work in *closed* monoidal categories, we extend the calculus with a new type former B/A and typing rules:

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

whose semantics is given by the corresponding structure of the closed monoidal category:

$$[\![\lambda x : A.\, t : B/A]\!] = abst([\![t]\!]) \qquad [\![t_1\ t_2]\!] = ev \cdot ([\![t_1]\!] \Box [\![t_2]\!])$$

where $abst : \mathscr{E}(C \Box A, B) \longrightarrow \mathscr{E}(C, B/A)$ is the natural isomorphism associated to the adjunction $(- \Box A) \dashv (-/A)$ and $ev : (B/A) \Box A \longrightarrow B$ is its counit. Jaskelioff and Moggi [2010] use the notation

B^A for this closed structure, while we use Lambek [1958]'s notation B/A to avoid the confusion with exponentials (which are right adjoints to cartesian products).

Other structures in $\mathscr{E}$ such as products and coproducts can be internalised in the metalanguages similarly. For example, for the cartesian product, we can add a type former $\times$ with typing rules:

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

Note that the first rule for $\langle t_1, t_2 \rangle$ introduces non-linearity to the syntax of the metalanguage: the variables in Γ may appear in both subterms t_1 and t_2.

Example 2.2. Let $\mathscr{E}$ be a closed monoidal category. If some type M together with two terms $(x : M, y : M \vdash \mu : M)$ and $(\cdot \vdash \eta : M)$ denotes a monoid in $\mathscr{E}$, then Cayley's theorem says that this monoid embeds into the monoid M/M with unit $(\cdot \vdash \lambda x.\, x : M/M)$ and multiplication

$$f : M/M,\ g : M/M \vdash \lambda x.\, f\ (g\ x) : M/M. \tag{10}$$

The embedding is given by $e = (x : M \vdash \lambda y.\, \mu : M/M)$, which is a monoid homomorphism. It has a left inverse $r = (f : M/M \vdash f\ \eta : M)$ such that $[\![r]\!] \cdot [\![e]\!] = \mathrm{id}_{[\![M]\!]}$. However, the inverse r is in general *not* a monoid homomorphism since $f(g\ \eta) \neq \mu[f\ \eta/x,\ g\ \eta/y]$.

This elementary result has surprisingly many applications in functional programming for optimisation, because the multiplication (10) 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 $\langle \mathbf{Set}, \times, 1 \rangle$, i.e. a list, this optimisation is known as *difference lists* [Hughes 1986]. When $\mathscr{E} = \langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$, this optimisation is known as *codensity transformation* [Hinze 2012]. We will also use this fact later for constructing modular models in Theorem 6.4.

Remark. We have not said anything about the equational laws for the type formers of the language. It is certainly possible to include such laws. For example, the β-law of products would be

$$\mathsf{let}\ (x_1, x_2) = (t_1, t_2)\ \mathsf{in}\ t_3 \quad \equiv \quad t_3[t_1/x_1, t_2/x_2].$$

Doing so would allow us to reason about monoidal categories inside the syntactic language, but a proper setup like this would take too much time before we get to the subject of this paper. Thus we will omit the equational laws and only use the language as a notational device for describing constructions in a monoidal category, and all reasoning is done semantically. For this reason, we only call this language a *metalanguage* rather than an *internal language* for monoidal categories.

3 EQUATIONAL SYSTEMS AND TRANSLATIONS

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, concrete examples of monoids in practice usually come with additional operations. For example, the state monad $(- \times S)^S$ comes with operations for reading and writing the mutable state, and the exception monad $- + E$ has operations for throwing and catching exceptions, and the (ordinary) monoid in **Set** of lists with concatenation has the operation of appending an element to a list.

Therefore we need a systematic way for talking about monoids equipped with additional *operations*, and also *equational theories* on these operations. Moreover, we wish to talk about combinations of such theories and their models, especially the *free models*, which are practically important since they play the role of abstract syntax of languages.

In this paper we use Fiore and Hur [2007, 2009]'s *equational systems* to formulate equational theories and use their results to construct free models, which we first recap below (§3.1). Then we extend their theory by introducing *translations* between equational systems, making them a category, and show some basic properties of colimits in this category (§3.4).

3.1 Equational Systems

An equational theory consists of the *signature* and *equations* of its operations. A concise way to specify a signature on a category $\mathscr{C}$ is just using a functor $\Sigma : \mathscr{C} \to \mathscr{C}$, and then a Σ-*algebra* is a pair of a *carrier* $A \in \mathscr{C}$ and a *structure map* $\alpha : \Sigma A \to A$. For example, the theory of semigroups has exactly a binary operation and one equation for associativity. Thus its signature functor is $\Sigma_{\mathrm{SG}} = - \,\square\, -$, and a Σ_{SG}-algebra is an object A equipped with an arrow $A \,\square\, A \to A$.

We denote the category of Σ-algebras by Σ-**Alg**, whose arrows from $\langle A, \alpha \rangle$ to $\langle B, \beta \rangle$ are *algebra homomorphisms*, i.e. arrows $h : A \to B$ 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 $\mathbf{U}_\Sigma : \Sigma\text{-}\mathbf{Alg} \to \mathscr{C}$ or just $\mathbf{U}$ when it is not ambiguous.

Equations of theories are usually presented as commutative diagrams (like the first diagram in (3) for associativity) containing formal arrows $\Sigma A \to A$ for the signature functor Σ. One way to formulate such a diagram is as a pair of functors $L, R : \Sigma\text{-}\mathbf{Alg} \to \Gamma\text{-}\mathbf{Alg}$ where the functor Γ encodes the starting node of the diagram, and L and R represent the two paths of the diagram.

Definition 3.1 (Fiore and Hur [2009])**.** An *equational system* $\hat{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ on a category $\mathscr{C}$ consists of four functors: (i) a *functorial signature* $\Sigma : \mathscr{C} \to \mathscr{C}$, (ii) a *functorial context* $\Gamma : \mathscr{C} \to \mathscr{C}$, and (iii) a pair of two *functorial terms* $L, R : \Sigma\text{-}\mathbf{Alg} \to \Gamma\text{-}\mathbf{Alg}$ such that $\mathbf{U}_\Gamma \circ L = \mathbf{U}_\Sigma$ and $\mathbf{U}_\Gamma \circ R = \mathbf{U}_\Sigma$.

An *algebra* or a *model* of $\hat{\Sigma}$ is a Σ-algebra $\langle A \in \mathscr{C}, \alpha : \Sigma A \to A \rangle$ such that $L\langle X, \alpha \rangle = R\langle X, \alpha \rangle$. The full subcategory of Σ-**Alg** containing all $\hat{\Sigma}$-algebras is denoted by $\hat{\Sigma}$-**Alg**.

Compared to alternative frameworks such as enriched algebraic theories [Kelly and Power 1993] or enriched Lawvere theories [Power 1999], equational systems are simpler and more flexible, yet still offer strong results for the existence of free algebras. Furthermore, we can use the metalanguage in §2.2 to specify the data for equational systems on monoidal categories $\mathscr{E}$ in a syntactic way (in fact, we were already doing this in Example 2.1 without mentioning it):

(i) To give a *functorial signature/context* $\mathscr{E} \to \mathscr{E}$, we add a distinguished base type τ to the metalanguage, and then every type expression T_τ in which τ occurs positively[1] induces a functor $[\![T_\tau]\!] : \mathscr{E} \to \mathscr{E}$ such that $[\![T_\tau]\!] A = [\![T_\tau]\!]_{\{\tau \mapsto A\}}$. The arrow mapping for the functor follows from the functoriality of the type constructors. Moreover, we add a new term syntax $T_\tau f$ to the metalanguage for the arrow mapping. It has the following typing rule:

$$\frac{x : A \vdash f : B \qquad \Gamma \vdash t : T_\tau[A/\tau]}{\Gamma \vdash (T_\tau f)\ t : T_\tau[B/\tau]}$$

For example, the type expression $T_\tau = \tau \,\square\, \tau$ denotes the functor $- \,\square\, - : \mathscr{E} \to \mathscr{E}$, and its arrow mapping is $\Gamma \vdash (T_\tau f)\ t : B \,\square\, B$ for all terms $A \vdash f : B$ and $\Gamma \vdash t : A \,\square\, A$.

(ii) To give a *functorial term* Σ-**Alg** $\to$ Γ-**Alg** for functors $\Sigma, \Gamma : \mathscr{E} \to \mathscr{E}$ denoted by type expressions S_τ and G_τ in the way mentioned above, we add a new primitive operation $\mathrm{op} : S_\tau \to \tau$ to the language. Then every term $x : G_\tau \vdash t : \tau$ induces a functor $T : \Sigma\text{-}\mathbf{Alg} \to \Gamma\text{-}\mathbf{Alg}$ such that

$$T\langle A, f : \Sigma A \to A \rangle = \langle A, [\![t]\!]_{\{\tau \mapsto A, \mathrm{op} \mapsto f\}} \rangle : \Sigma\text{-}\mathbf{Alg} \to \Gamma\text{-}\mathbf{Alg}$$

By structural induction on typing derivations in the style of Reynolds [1983]'s abstraction theorem, it is possible to show that $\mathbf{U}_\Gamma \circ T = \mathbf{U}_\Sigma$ as required in Definition 3.1. For example, the two terms (9) denote a pair of functorial terms $(- \,\square\, -)\text{-}\mathbf{Alg} \to (- \,\square\, - \,\square\, -)\text{-}\mathbf{Alg}$.

[1]The rule for positivity of a variable in a term is defined inductively as usual: τ occurs in itself positively and all other base types β both positively and negatively; and τ occurs in B/A positively (negatively) if τ occurs in B positively (negatively) and in A negatively (positively), and so on for other type formers.

Monoids. As an important example, the concept of monoids in a monoidal category $\langle \mathscr{E}, \square, I\rangle$ (§2.1) can be presented as an equational system when $\mathscr{E}$ has finite coproducts:

$$\mathbf{Mon} = (\Sigma_{\mathbf{Mon}} \triangleright \Gamma_{\mathbf{Mon}} \vdash L_{\mathbf{Mon}} = R_{\mathbf{Mon}}) \tag{11}$$

First we extend the metalanguage with a type constructor $A_1 + \cdots + A_n$ for finite products in $\mathscr{E}$, together with term syntax $[t_1, \ldots, t_n]$ for elimination and $\mathsf{inj}_i\ t$ for introduction. Then the functorial signature and context of the equational system **Mon** are given by type expressions

$$\Sigma_{\mathbf{Mon}} = (\tau \square \tau) + \mathsf{I} \qquad\qquad \text{and} \qquad\qquad \Gamma_{\mathbf{Mon}} = (\tau \square \tau \square \tau) + \tau + \tau,$$

and the pair of functorial terms $L_{\mathbf{Mon}} = R_{\mathbf{Mon}}$ are given by

$$\frac{\mathsf{op} : \tau \square \tau + \mathsf{I} \longrightarrow \tau}{x : (\tau \square \tau \square \tau) + \tau + \tau \vdash [l_1, l_2, l_3] \ = \ [r_1, r_2, r_3] : \tau}$$

where the components are as follows, c.f. the commutative diagrams (3):

$$\begin{gathered}
\mu = (x : \tau, y : \tau \vdash \mathsf{op}\ (\mathsf{inj}_1(x, y)) : \tau) \qquad\qquad \eta = (\vdash \mathsf{op}\ (\mathsf{inj}_2 *) : \tau) \\
l_1 = (x : \tau \square \tau \square \tau \vdash \mathsf{let}\ (x_1, x_2, x_3) = x \ \mathsf{in}\ \mu(\mu(x_1, x_2), x_3) : \tau) \\
r_1 = (x : \tau \square \tau \square \tau \vdash \mathsf{let}\ (x_1, x_2, x_3) = x \ \mathsf{in}\ \mu(x_1, \mu(x_2, x_3)) : \tau) \\
l_2 = (x : \tau \vdash \mu(\eta, x) : \tau) \qquad\qquad l_3 = (x : \tau \vdash \mu(x, \eta) : \tau) \\
r_2 = (x : \tau \vdash x \qquad : \tau) \qquad\qquad r_3 = (x : \tau \vdash x \qquad : \tau)
\end{gathered}$$

A model of the equational system **Mon** is exactly a monoid in $\langle \mathscr{E}, \square, I\rangle$.

Notation 3.2. As demonstrated above, although Definition 3.1 only considers exactly one functorial signature Σ and equation $\Gamma \vdash L = R$, multiple operations and equations can be expressed using coproducts. Given an equational system $\hat{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ on a category $\mathscr{C}$ with binary coproducts, we denote by $\hat{\Sigma} \mathrel{\urcorner_{\mathsf{op}}} \Sigma$ the extension of $\hat{\Sigma}$ with new operations of signature $\Sigma' : \mathscr{C} \longrightarrow \mathscr{C}$:

$$\hat{\Sigma} \mathrel{\urcorner_{\mathsf{op}}} \Sigma := (\Sigma + \Sigma' \ \ \triangleright \ \ \Gamma \ \ \vdash \ \ L \circ \pi = R \circ \pi)$$

where $\pi : (\Sigma + \Sigma')\text{-}\mathbf{Alg} \longrightarrow \Sigma\text{-}\mathbf{Alg}$ is the forgetful functor dropping Σ' operations. Similarly, we denote extending an equational system $\hat{\Sigma}$ with a new equation $\Gamma' \vdash L' = R'$ by

$$\hat{\Sigma} \mathrel{\urcorner_{\mathsf{eq}}} (\Gamma' \vdash L' = R') := (\Sigma \ \ \triangleright \ \ \Gamma + \Gamma' \ \ \vdash \ \ [L,\ L'] = [R,\ R']).$$

3.2 Free Algebras of Equational Systems

Among the algebras of an equational theory $\hat{\Sigma}$[2] 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:

$$\phi : \mathscr{C}(X, A) \ \cong \ \hat{\Sigma}\text{-}\mathbf{Alg}(\mathbf{Free}\, X, \langle A, \alpha\rangle)$$

Given any model $\langle A, \alpha\rangle$ of $\hat{\Sigma}$ and $g : X \longrightarrow A$, the morphism $\phi(g) : \mathbf{Free}\ X \longrightarrow \langle A, \alpha\rangle$ interprets the free algebra with the semantic model $\langle A, \alpha\rangle$. Fiore and Hur [2009] show various conditions for the existence of free algebras. In this paper, we will use the following one.

Theorem 3.3 (Fiore and Hur [2009])**.** *For all equational systems $\hat{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ over $\mathscr{C}$, if $\mathscr{C}$ is cocomplete and Σ and Γ preserve colimits of α-chains for a limit ordinal α, there are left adjoints*

$$\begin{tikzcd} \hat{\Sigma}\text{-}\mathbf{Alg} \arrow[r, hook, shift right] \arrow[r, phantom, "\perp"] & \Sigma\text{-}\mathbf{Alg} \arrow[l, shift right] \arrow[r, shift right] \arrow[r, phantom, "\perp"] & \mathscr{C} \arrow[l, shift right] \end{tikzcd} \tag{12}$$

to the inclusion functor $\hat{\Sigma}\text{-}\mathbf{Alg} \hookrightarrow \Sigma\text{-}\mathbf{Alg}$ and forgetful functor $\Sigma\text{-}\mathbf{Alg} \rightarrow \mathscr{C}$ respectively.

[2]Our convention is that we use the symbol $\hat{\Sigma}$ to mean Σ equipped with structure, and $\tilde{\Sigma}$ to mean $\hat{\Sigma}$ with more structure.

Notation 3.4. We denote the composite adjunction of (12) by $\mathbf{F}_{\hat{\Sigma}} \dashv \mathbf{U}_{\hat{\Sigma}} : \hat{\Sigma}\text{-}\mathbf{Alg} \to \mathscr{C}$, or simply $\mathbf{F} \dashv \mathbf{U}$ when $\hat{\Sigma}$ is understood. Moreover, the initial $\hat{\Sigma}$-algebra is denoted by $\langle \mu\hat{\Sigma}, \alpha^{\hat{\Sigma}} : \Sigma\mu\hat{\Sigma} \to \mu\hat{\Sigma} \rangle$.

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 informal 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 Γ does not preserve epimorphisms. The result of quotienting is the free $\hat{\Sigma}$-algebra.

Example 3.5. When $\mathscr{E}$ is cocomplete and $\Box : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ preserves α-chains for some limit ordinal α, then Theorem 3.3 is applicable to the equational system **Mon** (11). Moreover, when $\mathscr{E}$ is closed, there is a simple formula for free monoids: for every $A \in \mathscr{E}$, the free monoid over A is the initial algebra $\mu X.I + A \Box X$ equipped with appropriate monoid operations [Fiore 2008]. This formula is useful in practice: when $\mathscr{E}$ is $\langle \mathbf{Set}, \times, 1 \rangle$, it is exactly the usual definition $\mu X.1 + A \times X$ of A-lists; and when $\mathscr{E}$ is $\langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ or $\langle \mathbf{Endo}_\kappa(\mathbf{Set}), \star, \mathrm{Id} \rangle$ in §2.1, this gives formulas for free monads and free applicatives that are suitable for implementation [Rivas and Jaskelioff 2017].

3.3 Equational Systems for Monoids with Operations

Monoids equipped with additional operations are called Σ-*monoids* by Fiore et al. [1999]. Let $\Sigma : \mathscr{E} \to \mathscr{E}$ be a functor with a *pointed strength* θ, i.e. a natural transformation

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

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 [Fiore and Hur 2009, §7.2.1]. To denote Σ and θ syntactically, we extend the metalanguage with a type constructor Σ and the following typing rule

$$\frac{\cdot \vdash f : Y \qquad \Gamma \vdash t : (\Sigma X) \Box Y}{\Gamma \vdash \theta_{X, \langle Y, f \rangle}\; t : \Sigma(X \Box Y)}$$

Then the equational system Σ-**Mon** of Σ-*monoids* extends the theory **Mon** of monoids (11) with a new operation $\mathrm{op} : \Sigma\tau \to \tau$ and a new equation $L_{\Sigma\text{-}\mathbf{Mon}} = R_{\Sigma\text{-}\mathbf{Mon}}$:

$$\Sigma\text{-}\mathbf{Mon} = (\mathbf{Mon} \mathbin{\Lsh_{\mathrm{op}}} \Sigma) \mathbin{\Lsh_{\mathrm{eq}}} \big((\Sigma-) \Box - \vdash L_{\Sigma\text{-}\mathbf{Mon}} = R_{\Sigma\text{-}\mathbf{Mon}}\big) \tag{13}$$

where the new equation $L_{\Sigma\text{-}\mathbf{Mon}} = R_{\Sigma\text{-}\mathbf{Mon}}$ is given by

$$x : \Sigma\tau, y : \tau \vdash \mu(\mathrm{op}\; x, y) = \mathrm{op}((\Sigma\mu)(\theta_{\tau, \langle \tau, \eta \rangle}(x, y))) : \tau \tag{14}$$

The special case of (14) for $\Sigma = A \Box -$ with pointed strength $(A \Box X) \Box Y \cong A \Box (X \Box Y)$ is exactly *algebraicity* of operations $\mathrm{op} : A \Box \tau \to \tau$ on monoids τ used by Jaskelioff and Moggi [2010], which is a slight generalisation of Plotkin and Power [2001]'s definition of algebraicity for operations $(T-)^n \to T$ on a monad T. Thus we call (14) *generalised algebraicity*.

A model of the equational system Σ-**Mon** is called a Σ-monoid. When the monoidal category $\mathscr{E}$ is cocomplete and functors Σ, Γ, $\Box$ all preserve colimits of α-chains for some limit ordinal α, Theorem 3.3 ensures the existence of free Σ-monoids. When $\mathscr{E}$ is additionally closed, such as $\langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ in §2.1, there is again a simple description of the free Σ-monoid [Fiore and Hur 2007; Fiore and Saville 2017]: it is carried by the initial algebra $\mu X.I + A \Box X + \Sigma X$. 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].

Now let us look at some concrete examples. In all the following examples, the monoidal category $\langle \mathscr{E}, \square, I \rangle$ is assumed to have set-indexed 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:

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

Example 3.6 (Exception Throwing). Letting E be a set, the theory ET_E of *exception throwing* is the theory Σ_{ET_E}-**Mon** where $\Sigma_{\mathrm{ET}_E} = (\coprod_E 1) \square - : \mathscr{E} \to \mathscr{E}$, and $\coprod_E 1$ is the E-fold coproduct of the terminal object in $\mathscr{E}$ (which may be different from the monoidal unit I).

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

$$(\textstyle\coprod_E 1) \circ M = \coprod_E (1 \circ M) = \coprod_E 1 \quad \longrightarrow \quad M \tag{15}$$

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 $\mathscr{E} = \langle \mathbf{Endo}_\kappa(\mathbf{Set}), \star, \mathrm{Id} \rangle$, the theory describes applicative functors F with exception throwing:

$$(\textstyle\coprod_E 1) \star F \cong \coprod_E (1 \star F) = \coprod_E (\int^{a,b} 1a \times Fb \times -^{a \times b}) \cong \coprod_E (\int^{b} Fb) \quad \to \quad F \tag{16}$$

Note that exception throwing for monads (15) and for applicatives (16) 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.

Example 3.7 (Exception Catching). Although exception *throwing* is an algebraic operation, it is well known that *catching* is not: if we were to model it as an algebraic 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 the algebraicity (14) for $\Sigma M = M \times M$ implies that

$$ph : M \times M, k : M \vdash \mu(\mathit{catch}\; ph, k) = \mathit{catch}\,\langle \mu(\pi_1\; ph,\; k), \mu(\pi_2\; ph,\; k) \rangle : M \tag{17}$$

But this is undesirable because the *scopes of catching* are different: the left-hand side does not catch exceptions in k while the right-hand side catches exceptions in k.

Plotkin and Pretnar [2013]'s 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*.

An alternative view advocated by Wu et al. [2014] and Piróg et al. [2018] is that catching is also an operation of the theory of exceptions, albeit a more complex one which they call a *scoped operation*. This view allows one to construct free algebras of both throwing and catching, and then one can define different models/handlers of both catching and throwing [Yang et al. 2022].

Piróg et al. [2018]'s modelling of catching as a scoped operation can also be described as Σ_{Ec}-monoids in the monoidal category $\langle \mathbf{Endo}(\mathscr{C}), \circ, \mathrm{Id} \rangle$, where the signature functor $\Sigma_{\mathrm{Ec}} : \mathscr{E} \to \mathscr{E}$ is $\Sigma_{\mathrm{Ec}} = (1 \circ -) + (\mathrm{Id} \times \mathrm{Id}) \circ - \circ -$ with the pointed strength $\theta_{X, \langle Y, f \rangle}$ for all $X \in \mathscr{E}$ and $\langle Y, f \rangle : I/\mathscr{E}$:

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

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 3.6, and the second operation

$$\mathit{catch} : (\mathrm{Id} \times \mathrm{Id}) \circ M \circ M \cong (M \times M) \circ M \quad \longrightarrow \quad M \tag{18}$$

is catching. The trick here to avoid the undesirable equation (17) 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]: $catch\ (\langle p, h \rangle, k)$ is understood as handling the exception in p with h and then continuing as k. Then the generalised algebraicity (14) instantiates to

$$ph : M \times M, k : M, k' : M \vdash \mu(catch\ (ph, k), k') = catch\ (ph, \mu(k, k')) : M \tag{19}$$

Unlike (17), 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.

Additionally, 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:

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

where $\eta : \mathsf{I} \longrightarrow \tau$, $throw : 1 \longrightarrow \tau$, and $catch : (\tau \times \tau) \mathbin{\square} \tau \longrightarrow \tau$. These equations can be alternatively presented with an empty context by replacing all the k's with η as in $\cdot \vdash\ catch(\langle throw, \eta \rangle, \eta) = \eta : \tau$, which is equivalent to the first equation above, since by (19), $catch(\langle x, y \rangle, \eta); k = catch(\langle x, y \rangle, k)$.

Example 3.8. Let S be a set. The theory 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 $((\prod_S \mathsf{I}) \mathbin{\square} \tau) + ((\coprod_S \mathsf{I}) \mathbin{\square} \tau)$, whose first component represents an operation $g : (\prod_S \mathsf{I}) \mathbin{\square} \tau \longrightarrow \tau$ reading the state, and the second component represents an operation $p : (\coprod_S \mathsf{I}) \mathbin{\square} \tau \longrightarrow \tau$ writing an S-value into the state. Plotkin and Power [2002]'s 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

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

where $p_s(x)$ abbreviates $p(\mathsf{inj}_s *, x)$.

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].

3.4 Functorial Translations

Arrows between equational systems are not studied in the work by Fiore and Hur [2007, 2009], but we need them later for talking about combinations of equational systems. A natural idea for arrows from an equational system $\hat{\Sigma}$ to another $\hat{\Gamma}$ is a *translation* from operations in $\hat{\Sigma}$ to *terms* of $\hat{\Gamma}$, preserving equations in a suitable sense. However, a technical difficulty is that equational systems $\hat{\Gamma}$ may not have terms, i.e. initial algebras. In the following, we avoid this by introducing a more abstract and simpler definition which we call *functorial translations* between equational systems. And they seem to be the right notion of arrows between equational systems.

Definition 3.9. A *functorial translation* of equational systems on $\mathscr{C}$ from $\hat{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ to $\hat{\Sigma}' = (\Sigma' \triangleright \Gamma' \vdash L' = R')$ is a functor $T : \hat{\Sigma}'\text{-}\mathbf{Alg} \longrightarrow \hat{\Sigma}\text{-}\mathbf{Alg}$ such that $\mathbf{U}_{\hat{\Sigma}} \circ T = \mathbf{U}_{\hat{\Sigma}'}$, where $\mathbf{U}_{\hat{\Sigma}} : \hat{\Sigma}\text{-}\mathbf{Alg} \longrightarrow \mathscr{C}$ and $\mathbf{U}_{\hat{\Sigma}'} : \hat{\Sigma}'\text{-}\mathbf{Alg} \longrightarrow \mathscr{C}$ are the forgetful functors. Equational systems on $\mathscr{C}$ and translations form a category $\mathbf{Eqs}(\mathscr{C})$, in which the identity arrows are the identity functors $\hat{\Sigma}\text{-}\mathbf{Alg} \longrightarrow \hat{\Sigma}\text{-}\mathbf{Alg}$, and composition of translations $T \circ T'$ is functor composition.

Note the contravariance in the definition: a translation $\hat{\Sigma} \longrightarrow \hat{\Sigma}'$ is a functor $\hat{\Sigma}'\text{-}\mathbf{Alg} \longrightarrow \hat{\Sigma}\text{-}\mathbf{Alg}$ from the opposite direction preserving carriers and homomorphisms, since $\mathbf{U}_{\hat{\Sigma}} \circ T = \mathbf{U}_{\hat{\Sigma}'}$.

Example 3.10. The theory **Grp** of groups in a category $\mathscr{C}$ with finite coproducts and products is the theory **Mon** of monoids in $\langle \mathscr{C}, \times, 1 \rangle$ extended with a unary inverse operation:

$$\mathbf{Grp} = (\mathbf{Mon} \urcorner_{\mathrm{op}} -) \urcorner_{\mathrm{eq}} (x : \tau \vdash \mu\langle x, x^{-1}\rangle) = \eta : \tau)$$

where x^{-1} denote the newly added operation. Then there is a translation $T : \mathbf{Mon} \to \mathbf{Grp}$ that maps every $\langle X, \alpha : (\Sigma_{\mathbf{Mon}} X + X) \to X \rangle$ in **Grp-Alg** to an object $\langle X, \alpha \cdot \iota_1 : \Sigma_{\mathbf{Mon}} X \to X \rangle$ in **Mon-Alg** by forgetting the newly added operation. In the rest of the paper, we call translations like $T : \mathbf{Mon} \to \mathbf{Grp}$ that simply forgets some operations and equations *inclusion translations.*

Relative Free Algebras. Let $\mathbf{Eqs}_c(\mathscr{C})$ be the full subcategory of $\mathbf{Eqs}(\mathscr{C})$ containing equational systems whose functorial signature and context preserve colimits of all α-chains for some limit ordinal α. By Theorem 3.3, every equational system $\hat{\Sigma}$ in $\mathbf{Eqs}_c(\mathscr{C})$ has the free-forgetful adjunction $\mathbf{F}_{\hat{\Sigma}} \dashv \mathbf{U}_{\hat{\Sigma}} : \hat{\Sigma}\text{-}\mathbf{Alg} \to \mathscr{C}$ when $\mathscr{C}$ is cocomplete. In this case, functorial translations are equivalent to the traditional notion of translations as monad morphisms. Proofs of the results in this paper can be found in the supplemented appendices.

Lemma 3.11. *For cocomplete $\mathscr{C}$ and $\hat{\Sigma}, \hat{\Gamma} \in \mathbf{Eqs}_c(\mathscr{C})$, functorial translations $T : \hat{\Sigma} \to \hat{\Gamma}$ are in bijection with monad morphisms $m : \mathbf{U}_{\hat{\Sigma}}\mathbf{F}_{\hat{\Sigma}} \to \mathbf{U}_{\hat{\Gamma}}\mathbf{F}_{\hat{\Gamma}}$.*

In the adjunction $\mathbf{F}_{\hat{\Sigma}} \dashv \mathbf{U}_{\hat{\Sigma}} : \hat{\Sigma}\text{-}\mathbf{Alg} \to \mathscr{C}$, the category $\mathscr{C}$ can be viewed as the category $\emptyset$-**Alg** for the empty theory $\emptyset$ with no operations, and $\mathbf{U}_{\hat{\Sigma}} : \hat{\Sigma}\text{-}\mathbf{Alg} \to \emptyset\text{-}\mathbf{Alg}$ is the unique translation from $\emptyset$ to $\hat{\Sigma}$. The fact that $\mathbf{U}_{\hat{\Sigma}}$ always has left adjoint can be generalised to any functorial translations.

Theorem 3.12. *For cocomplete $\mathscr{C}$, every functorial translation $T : \hat{\Sigma} \to \hat{\Gamma}$ in $\mathbf{Eqs}_c(\mathscr{C})$ as a functor $T : \hat{\Gamma}\text{-}\mathbf{Alg} \to \hat{\Sigma}\text{-}\mathbf{Alg}$ has a left adjoint $F : \hat{\Sigma}\text{-}\mathbf{Alg} \to \hat{\Gamma}\text{-}\mathbf{Alg}$.*

Proof sketch. The free $\hat{\Gamma}$-algebra over a $\hat{\Sigma}$-algebra $\langle A, \alpha \rangle$ are constructed as the initial algebra of the equational system $\hat{\Gamma}$ extended with A as constants and equations saying that Σ-operations on A constants are exactly α. A detailed proof can be found in the supplemented appendix. □

For example, Theorem 3.12 applied to the translation $\mathbf{Mon} \to \mathbf{Grp}$ in Example 3.10 constructs free groups over monoids. This theorem will later be used for constructing *free modular models.*

Colimits. Colimits in $\mathbf{Eqs}(\mathscr{C})$ allow one to 'glue' equational systems. 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 $L \rightrightarrows \mathbf{Grp} + \mathbf{Mon}$ encoding the interaction of operations.

Theorem 3.13. *The category $\mathbf{Eqs}_c(\mathscr{C})$ is cocomplete if $\mathscr{C}$ is cocomplete.*

Proof sketch. It is sufficient to show the existence of arbitrary coproducts and coequalisers: coproducts are defined by taking the coproduct of functorial signatures and functorial context; coequalisers are defined by adding a new equation. The appendix provides a more detailed proof. □

Lastly, the following direct description of a special case of pushouts is sometimes convenient.

Lemma 3.14. *Let $\hat{\Sigma} \in \mathbf{Eqs}(\mathscr{C})$ for $\mathscr{C}$ a category with finite coproducts, and for $i \in \{1, 2\}$, let $\Phi_i : \mathscr{C} \to \mathscr{C}$ be a functorial signature and $E_i = (\Psi_i \vdash L_i = R_i)$ be an equation. 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:*

$$\begin{tikzcd}
\hat{\Sigma} \arrow[r, "T_2"] \arrow[d, "T_1"'] & \hat{\Sigma} \urcorner_{\mathrm{op}} \Phi_2 \urcorner_{\mathrm{eq}} E_2 \arrow[d] \\
\hat{\Sigma} \urcorner_{\mathrm{op}} \Phi_1 \urcorner_{\mathrm{eq}} E_1 \arrow[r] & \hat{\Sigma} \urcorner_{\mathrm{op}} (\Phi_1 + \Phi_2) \urcorner_{\mathrm{eq}} E
\end{tikzcd}$$

where $E = (\Psi_1 + \Psi_2 \vdash [L_1 \circ \alpha_1, L_2 \circ \alpha_2] = [R_1 \circ \alpha_1, R_2 \circ \alpha_2])$ and $\alpha_i : (\Sigma + (\Phi_1 + \Phi_2))\text{-}\mathbf{Alg} \to (\Sigma + \Phi_i)\text{-}\mathbf{Alg}$ is the projection functor.

4 MONOIDAL THEORY FAMILIES

As motivated in §1 and §2, we are primarily interested in equational theories that extend the theory **Mon** of monoids with more operations. The more precise way to say it now is that we are interested in the coslice category $\mathbf{Mon}/\mathbf{Eqs}_c(\mathscr{E})$. However, some operations are clearly more complex than others: exception catching as a scoped operation is more complex than exception throwing, and λ-abstraction as a variable-binding operation is more complex than λ-application. Thus in this section, we do a finer classification of theories in $\mathbf{Mon}/\mathbf{Eqs}_c(\mathscr{E})$, grouping them into different *monoidal theory families* according to the complexity of their operations. It turns out that the simplest kind of operations, the algebraic ones, play a special role among all operations.

Definition 4.1. A *monoidal theory family* over a monoidal category $\mathscr{E}$ is a full subcategory $\mathcal{F} \subseteq \mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$ of the coslice category of equational systems under the theory **Mon** of monoids such that (i) the collection of objects of $\mathcal{F}$ is closed under finite coproducts in $\mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$, and (ii) each $\langle \hat{\Sigma}, T\rangle \in \mathcal{F}$ has free algebras $\mathscr{E} \to \hat{\Sigma}\text{-}\mathbf{Alg}$.

Note that the coproducts in $\mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$ are equivalently pushouts $\hat{\Sigma} \leftarrow \mathbf{Mon} \to \hat{\Gamma}$ in $\mathbf{Eqs}(\mathscr{E})$, so coproducts in $\mathcal{F}$ are intuitively combining theories while identifying their monoid operations. This definition itself is not very interesting, but its examples and their connections are interesting. The examples below assume a monoidal category $\mathscr{E}$ with the following properties.

Definition 4.2. A monoidal category $\mathscr{E}$ is called *cocordial* when its product $\Box : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ is **co**continuous (i.e. $\Box$ preserves colimits of α-chains for some limit ordinal α) and $\mathscr{E}$ is a **co**complete and **r**ight **di**stributive (i.e. $\Box$ preserves coproducts in the first argument) monoid**al** category.

The monoidal categories introduced in §2.1, $\langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id}\rangle$ for an lκp $\mathscr{C}$, $\langle \mathscr{C}, \times, 1\rangle$ for a cocomplete cartesian $\mathscr{C}$, $\langle \mathbf{Endo}_\kappa(\mathbf{Set}), \star, \mathrm{Id}\rangle$, and $\langle \mathbf{Endo}_f(\mathbf{Set})^{\mathbb{G}}, \star, I\rangle$ are all cocordial.

Algebraic Operations. Our first example is the family $\mathbf{ALG}(\mathscr{E})$ of *algebraic operations* on a cocordial category $\langle \mathscr{E}, \Box, I\rangle$. The full subcategory $\mathbf{ALG}(\mathscr{E}) \subseteq \mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$ contains

$$\{\langle \Sigma\text{-}\mathbf{Mon} \,{}^\lnot\!\!\!\rceil_{\mathrm{eq}}\, E,\ T\rangle \ \mid\ A \in \mathscr{E},\ \Sigma = A \Box -,\ E = (\mathbf{K}_B \vdash L = R)\} \tag{20}$$

where $T : \mathbf{Mon} \hookrightarrow \Sigma\text{-}\mathbf{Mon} \,{}^\lnot\!\!\!\rceil_{\mathrm{eq}}\, E$ is the inclusion translation, and E is an arbitrary functorial equation whose context is a *constant functor* (see the remark below). In other words, $\mathbf{ALG}(\mathscr{E})$ contains all equational systems Σ-**Mon** (13) extended with an equation for some $\Sigma = A \Box -$ with the canonical pointed strength $(A \Box X) \Box Y \cong A \Box (X \Box Y)$.

The category $\mathbf{ALG}(\mathscr{E})$ satisfies the conditions in Definition 4.1 since it is closed under coproducts following Lemma 3.14, and all equational systems (20) in $\mathbf{ALG}(\mathscr{E})$ have free algebras by Theorem 3.3, since the signature and context functor of (20) preserve colimits of α-chains for some ordinal α.

In particular the theory of exceptions (Example 3.6) and states (Example 3.8) are in $\mathbf{ALG}(\mathscr{E})$. When $\mathscr{E} = \langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id}\rangle$ for an lκp category $\mathscr{C}$, $\mathbf{ALG}(\mathscr{E})$ consists of theories of algebraic operations $A \circ M \to M$ for $A \in \mathbf{Endo}_\kappa(\mathscr{C})$ on κ-accessible monads M. When $\mathscr{E}$ is $\langle \mathbf{Endo}_\kappa(\mathbf{Set}), \star, \mathrm{Id}\rangle$, it then contains theories of applicatives F with 'applicative-algebraic' operations $A \star F \to F$.

The family $\mathbf{ALG}(\mathscr{E})$ of theories of algebraic operations is closely related to traditional notions of (presentations of) equational theories and syntactic translations between them.

Proposition. *The category* $\mathbf{ALG}(\langle \mathbf{Endo}_f(\mathbf{Set}), \circ, \mathrm{Id}\rangle)$ *is equivalent to the category of (presentations of) first-order equational theories and their syntactic translations (see e.g. [Fiore and Mahmoud 2014] for a detailed definition). Moreover, the category of* $\mathbf{ALG}(\langle \mathbf{Endo}_f(\mathbf{Set})^{\mathbb{G}}, \star, I\rangle)$ *is equivalent to the category of presentations of graded algebraic theories and their morphisms introduced by Kura [2020].*

We forgo a proof here. The key observation for showing this is that for any cocordial $\mathscr{E}$ and $A, B \in \mathscr{E}$, translations $(A \square -)$-**Mon** $\to (B \square -)$-**Mon** in $\mathbf{ALG}(\mathscr{E})$ are in bijection with arrows $A \to B^*$ in $\mathscr{E}$ where B^* is the free monoid over B in $\mathscr{E}$.

Theorem 4.3. *For a cocordial monoidal category $\mathscr{E}$, there is an equivalence $\mathbf{ALG}(\mathscr{E}) \cong \mathbf{Mon}(\mathscr{E})$ between the category $\mathbf{ALG}(\mathscr{E})$ and the category $\mathbf{Mon}(\mathscr{E})$ of monoids in $\mathscr{E}$.*

Proof sketch. Every $\tilde{\Sigma} \in \mathbf{ALG}(\mathscr{E})$ is mapped to its initial algebra treated as a monoid. Every monoid M is mapped to the theory of M*-actions on monoids*. The appendix provides a detailed proof. □

Instantiating $\mathscr{E}$ with $\langle \mathbf{Endo}_f(\mathscr{C}), \circ, \mathrm{Id}\rangle$ for an lfp $\mathscr{C}$, we obtain the classical correspondence between finitary monads and (presentations of) first-order equational theories. What is new is that Theorem 4.4 is applicable to other cocordial monoidal categories in §2.1, giving us equivalences of cartesian monoids/applicative functors/graded monads and the corresponding categories of theories of algebraic operations. This general monoid-theory correspondence seems new to us.

Another interesting property of $\mathbf{ALG}(\mathscr{E})$ is the following saying that almost all equational theories of operations on monoids can be turned into one in $\mathbf{ALG}(\mathscr{E})$ 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.

Theorem 4.4. *Let $\mathscr{E}$ be a cocordial monoidal category. (i) The category $\mathbf{ALG}(\mathscr{E})$ is a coreflective subcategory of $\mathbf{Mon}/\mathbf{Eqs}_c(\mathscr{E})$, i.e. there is an adjunction $\mathbf{ALG}(\mathscr{E}) \underset{\top}{\leftrightarrows} \mathbf{Mon}/\mathbf{Eqs}_c(\mathscr{E})$. (ii) Moreover, the coreflector $\lfloor - \rfloor$ preserves initial algebras: for every $\langle \hat{\Sigma} \in \mathbf{Eqs}_c(\mathscr{E}), T : \mathbf{Mon} \to \hat{\Sigma}\rangle$, the initial $\hat{\Sigma}$-algebra (viewed as a monoid using T) is isomorphic to the initial algebra of $\lfloor \langle \hat{\Sigma}, T\rangle \rfloor$ as monoids.*

Proof sketch. Every $\langle \hat{\Sigma}, T\rangle \in \mathbf{Mon}/\mathbf{Eqs}_c(\mathscr{E})$ has an initial algebra $\mu\hat{\Sigma}$, which has a monoid structure under the translation T. The coreflector maps every $\langle \hat{\Sigma}, T\rangle$ to the theory of $\mu\hat{\Sigma}$-**Act** of $\mu\hat{\Sigma}$-actions. The category of algebras of $\mu\hat{\Sigma}$-**Act** is equivalent to the coslice category $T\mu\hat{\Sigma}/\mathbf{Mon}(\mathscr{E})$, so the initial algebra $\mu\hat{\Sigma}$-**Act** is still the same monoid $T\mu\hat{\Sigma}$. The appendix provides more details. □

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

Scoped Operations. Our next example of monoidal theory families is the family $\mathbf{SCP}(\mathscr{E})$ of *scoped (and algebraic) operations*, such as exception catching (Example 3.7). The family $\mathbf{SCP}(\mathscr{E})$ is given by the full subcategory of $\mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$ containing objects

$$\{\langle \Sigma\text{-}\mathbf{Mon} \urcorner_{\mathrm{eq}} E,\ T\rangle \mid A, B \in \mathscr{E},\ \Sigma = (A \square - \square -) + (B \square -),\ E = (\mathbf{K}_C \vdash L = R)\} \tag{21}$$

where $T : \mathbf{Mon} \to \Sigma\text{-}\mathbf{Mon} \urcorner_{\mathrm{eq}} E$ is the inclusion translation. The pointed strength of Σ needed in the definition of Σ-**Mon** (13) is as follows, where the boxed Y is inserted using $\eta^Y : I \to Y$:

$$(\Sigma X) \square Y \cong (A \square X \square X \square Y) + (B \square X \square Y) \to (A \square X \boxed{\square Y} \square X \square Y) + (B \square X \square Y) \cong \Sigma(X \square Y).$$

Piróg et al. [2018] introduced scoped operations to model non-algebraic operations on monads that delimit scopes. As explained in Example 3.7, the trick is to let the operation take an explicit continuation. Indeed, operations $f : A \square M \square M \to M$ on a monoid M satisfying generalised algebraicity (14) are in bijection with arrows $g : A \square M \to M$ *without algebraicity*:

$$f \mapsto (A \square M \xrightarrow{A \square M \square \eta^M} A \square M \square M \xrightarrow{f} M) \qquad g \mapsto (A \square M \square M \xrightarrow{A \square \mu^M} A \square M \xrightarrow{g} M)$$

A direct corollary of Theorem 4.4 is that the initial-algebra preserving coreflection there restricts to $\mathbf{ALG}(\mathscr{E}) \underset{\top}{\leftrightarrows} \mathbf{SCP}(\mathscr{E})$. Thus the terms of 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.

Variable-Binding Operations. Our final example is the monoidal theory family of *variable-binding operations* studied by Fiore et al. [1999]. For now we work concretely in the monoidal category $\langle \mathbf{Set}^{\mathbf{Fin}}, \bullet, V \rangle$ (5), but it is possible to replace $\mathbf{Set}^{\mathbf{Fin}}$ with $\mathbf{Endo}_\kappa(\mathbf{Set})$ for infinitary syntax.

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

A binding signature then determines an endofunctor $\Sigma_{\langle O,a \rangle} : \mathbf{Set}^{\mathbf{Fin}} \to \mathbf{Set}^{\mathbf{Fin}}$:

$$\Sigma_{\langle O,a \rangle} = \coprod_{o \in O,\ a(o) = \langle n_i \rangle_{1 \leqslant i \leqslant k}} \prod_{1 \leqslant i \leqslant k} (-)^{V^{n_i}} \tag{22}$$

where $(-)^{V^{n_i}}$ is the exponential by n_i-fold product of the monoidal unit V. The monoidal theory family $\mathbf{VAR}(\mathbf{Set}^{\mathbf{Fin}}) \subseteq \mathbf{Mon}/\mathbf{Eqs}(\mathbf{Set}^{\mathbf{Fin}})$ then contains objects

$$\{ \langle \Sigma_{\langle O,a \rangle}\text{-}\mathbf{Mon} \Lsh_{\text{eq}} E,\ T \rangle \mid O \text{ a set}, a : O \to \mathbb{N}^*, E = (\mathbf{K}_B \vdash L = R) \}$$

where $T : \mathbf{Mon} \to \Sigma_{\langle O,a \rangle}\text{-}\mathbf{Mon} \Lsh_{\text{eq}} E$ is the inclusion translation. The definition of $\mathbf{VAR}(\mathbf{Set}^{\mathbf{Fin}})$ satisfies Definition 4.1 because it is closed under coproducts by Lemma 3.14, and the functorial signature $\Sigma_{\langle O,a \rangle}$ and context $\mathbf{K}_B$ are finitary. The finitariness of $\Sigma_{\langle O,a \rangle}$ is a consequence of $(-)^V$ being a left adjoint to the right Kan extension $\mathbf{Ran}_{V+1}$, so $(-)^V$ preserves all colimits.

Again, the coreflector in Theorem 4.4 allows us to turn every theory in $\mathbf{VAR}$ into one with only algebraic operations but has 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).

Summary. It is a good time to reflect what we have so far: (i) we have seen how to present equational systems, in particular theories of monoids with operations, using the metalanguage (§3.1); (ii) we can build (relative) free models (Theorem 3.3 and 3.12); (iii) we can modularly *combine* such theories using colimits (§3.4); (iv) these theories are classified into families (§4), with the family of algebraic operations playing a special role (Theorem 4.3 and 4.4).

5 MODULAR MODELS OF MONOIDS

Now we come to the second part of this paper on semantic modularity as motivated in §1. The main definition is *modular models* of a theory $\tilde{\Sigma}$ in a monoidal theory family $\mathcal{F}$. We have two equivalent formulations, one based on $\mathbf{CAT}$-*valued functors* (Definition 5.2), and another based on *split fibrations* (Theorem 5.4). We also explain how modular models are used to interpret the abstract syntax of an equational system (§5.2) in a modular way.

5.1 Modular Models

Notation 5.1. Given a monoidal theory family $\mathcal{F}$, each $\tilde{\Sigma} \in \mathcal{F}$ is pair $\langle \hat{\Sigma}, T \rangle$ of an equational system $\hat{\Sigma}$ and a translation $\mathbf{Mon} \to \hat{\Sigma}$. We use the notation $\tilde{\Sigma}$-$\mathbf{Alg}$ to mean the category $\hat{\Sigma}$-$\mathbf{Alg}$ of algebras for the underlying equational system $\hat{\Gamma}$ of $\tilde{\Sigma}$.

Definition 5.2. Given a monoidal theory family $\mathcal{F}$, a *modular model* M of some $\tilde{\Sigma} \in \mathcal{F}$ is a family of functors $M_{\tilde{\Sigma}} : \tilde{\Sigma}\text{-}\mathbf{Alg} \to (\tilde{\Sigma} + \tilde{\Gamma})\text{-}\mathbf{Alg}$ for each $\tilde{\Sigma} \in \mathcal{F}$ together with a family of natural transformations $M_T : M_{\tilde{\Sigma}} \circ T \longrightarrow (T + \tilde{\Gamma}) \circ M_{\tilde{\Sigma}'}$ for each translation $T : \tilde{\Sigma} \longrightarrow \tilde{\Sigma}'$ in $\mathcal{F}$:

$$\begin{tikzcd}
\tilde{\Sigma}\text{-}\mathbf{Alg} \arrow[d, "M_{\tilde{\Sigma}}"'] \arrow[dr, phantom, "\overset{M_T}{\Longrightarrow}"] & \tilde{\Sigma}'\text{-}\mathbf{Alg} \arrow[l, "T"'] \arrow[d, "M_{\tilde{\Sigma}'}"] \\
(\tilde{\Sigma} + \tilde{\Gamma})\text{-}\mathbf{Alg} & (\tilde{\Sigma}' + \tilde{\Gamma})\text{-}\mathbf{Alg} \arrow[l, "T + \tilde{\Gamma}"]
\end{tikzcd}$$

such that M_{id} is the identity natural transformation, and for all $T : \tilde{\Sigma} \longrightarrow \tilde{\Sigma}'$ and $T' : \tilde{\Sigma}' \longrightarrow \tilde{\Sigma}''$, the square $M_{T' \circ T}$ is exactly the pasting of M_T and $M_{T'}$, i.e. $M_{T' \circ T} = ((T + \tilde{\Gamma}) \circ M_{T'}) \cdot (M_T \circ T')$. The modular model M is called *strict* when the natural transformations M_T are identities.

Definition 5.2 is based on *lax transformations* of **CAT**-valued functors. **CAT**-valued functors are equivalent to *split fibrations* via the *Grothendieck construction*, and it turns out we can alternatively formulate modular models based on fibrations. The fibrational formulation is not as compact as Definition 5.2, but is usually easier to work with, especially when thinking about certain 'dependently typed' constructions such as mapping each $\tilde{\Sigma}$ in $\mathcal{F}$ to the initial algebra in $\tilde{\Sigma}$-**Alg**. Thus we show the fibrational formulation before diving into any examples of modular models.

We will only need the very basics about fibrations (see e.g. Jacobs [1999, Chapter 1]), and we will give explicit descriptions of constructions for the reader unfamiliar with fibrations. For completeness, a summary of concepts about fibration that we use in this paper is in the appendix.

For every monoidal theory family $\mathcal{F}$, the **CAT**-valued functor $(-)$-**Alg** $: \mathcal{F}^{op} \longrightarrow$ **CAT** induces a category $\mathcal{F}$-**Alg** and a split fibration $\mathcal{F}$-**Alg** $\longrightarrow \mathcal{F}$, which we explicitly describe below. The intuition is that $\mathcal{F}$-**Alg** is the category of *all models of all equational systems* in $\mathcal{F}$.

Definition 5.3. For every monoidal theory family $\mathcal{F}$, the objects of category $\mathcal{F}$-**Alg** are tuples

$$\langle \hat{\Sigma} \in \mathbf{Eqs}(\mathscr{E}), \quad T_\Sigma : \mathbf{Mon} \longrightarrow \hat{\Sigma}, \quad A \in \mathscr{E}, \quad \alpha : \Sigma A \longrightarrow A \rangle$$

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

$$\begin{tikzcd}
\Sigma A \arrow[r, "\alpha"] \arrow[d, "\Sigma h"'] & A \arrow[d, "\boxed{h}"] & \text{(in } \hat{\Sigma}\text{-}\mathbf{Alg}) & & & \\
\Sigma B \arrow[r, "T\langle B,\beta\rangle"'] & B & & & \Gamma B \arrow[r, "\beta"] \arrow[lll, "\boxed{T}"'] & B \quad \text{(in } \hat{\Gamma}\text{-}\mathbf{Alg})
\end{tikzcd}$$

The identities arrows are pairs of identity translations and homomorphisms: $\langle \mathrm{Id} : \hat{\Sigma} \longrightarrow \hat{\Sigma}, \mathrm{id} : A \longrightarrow A \rangle$. The composition of two arrows $\langle T, h \rangle$ and $\langle T', h' \rangle$ are $\langle T \circ T', h \cdot h' \rangle$.

The split fibration $P : \mathcal{F}$-**Alg** $\longrightarrow \mathcal{F}$ is the evident projection: $P\langle \hat{\Sigma}, T_\Sigma, A, \alpha \rangle = \langle \hat{\Sigma}, T_\Sigma \rangle$ and $P\langle T, h \rangle = T$. It is equipped with a split cleavage sending arrows $T : \langle \hat{\Sigma}, T_\Sigma \rangle \longrightarrow \langle \hat{\Gamma}, T_\Gamma \rangle \in \mathcal{F}$ and objects $\langle \hat{\Gamma}, T_\Gamma, B, \beta \rangle \in \mathcal{F}$-**Alg** to $\langle T, \mathrm{id} \rangle : \langle \hat{\Sigma}, T_\Sigma, B, T\langle B, \beta \rangle \rangle \longrightarrow \langle \hat{\Gamma}, T_\Gamma, B, \beta \rangle$.

Given an equational system $\tilde{\Gamma} \in \mathcal{F}$, we are also interested in models of equational systems in $\mathcal{F}$ that are additionally equipped with a $\tilde{\Gamma}$-algebra. Such $(\mathcal{F} + \tilde{\Gamma})$*-algebras* can be obtained by a *change-of-base* for the fibration $P : \mathcal{F}$-**Mon** $\longrightarrow \mathcal{F}$ along the functor $(- + \tilde{\Gamma}) : \mathcal{F} \longrightarrow \mathcal{F}$, which is the following pullback in the category **CAT** of categories:

$$\begin{tikzcd}
(\mathcal{F} + \tilde{\Gamma})\text{-}\mathbf{Alg} \arrow[r, "K"] \arrow[d, "Q"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{F}\text{-}\mathbf{Alg} \arrow[d, "P"] \\
\mathcal{F} \arrow[r, "-+\tilde{\Gamma}"'] & \mathcal{F}
\end{tikzcd} \tag{23}$$

Explicitly, the objects of $(\mathcal{F} + \tilde{\Gamma})$-**Alg** are tuples:

$$\langle \hat{\Sigma} \in \mathbf{Eqs}(\mathscr{E}), \quad T_\Sigma : \mathbf{Mon} \to \hat{\Sigma}, \quad A \in \mathscr{E}, \quad \alpha : \Sigma A \to A, \quad \beta : \Gamma A \to A \rangle$$

such that $\langle \hat{\Sigma}, T_\Sigma \rangle \in \mathcal{F}$, $\langle A, \alpha \rangle \in \hat{\Sigma}$-**Alg**, $\langle A, \beta \rangle \in \hat{\Gamma}$-**Alg**, and $T_\Gamma \langle A, \alpha \rangle = T_\Sigma \langle A, \beta \rangle \in \mathbf{Mon}(\mathscr{E})$. Arrows in $(\mathcal{F}+\hat{\Gamma})$-**Alg** are similar to those $\langle T, h \rangle$ in $\mathcal{F}$-**Alg**, but require h also to be a $\hat{\Gamma}$-homomorphism.

The pullback (23) also induces two functors Q and K. The functor $Q : (\mathcal{F} + \tilde{\Gamma})\text{-}\mathbf{Mon} \to \mathcal{F}$ is the projection to $\mathcal{F}$, and it is a split fibration with a split cleavage similar to that of P. The functor $K : (\mathcal{F} + \hat{\Gamma})\text{-}\mathbf{Mon} \to \mathcal{F}\text{-}\mathbf{Mon}$ maps objects $\langle \hat{\Sigma}, T_\Sigma, A, \alpha, \beta \rangle$ to $\langle \langle \hat{\Sigma}, T_\Sigma \rangle + \langle \hat{\Gamma}, T_\Gamma \rangle, A, [\alpha, \beta] \rangle$. Additionally, the pair $\langle K, - + \hat{\Gamma} \rangle$ is a split fibration morphism from Q to P.

Now we have enough machinery to spell out the fibrational formulation of modular models.

Theorem 5.4. *Modular models M of some $\tilde{\Gamma} \in \mathcal{F}$ as in Definition 5.2 are in bijection with functors $\bar{M} : \mathcal{F}\text{-}\mathbf{Alg} \to (\mathcal{F} + \tilde{\Gamma})\text{-}\mathbf{Alg}$ such that $Q \circ \bar{M} = P$ with P and Q as in (23). A modular model M is strict iff $\bar{M}$ is a split fibration morphism from P to Q, that is, $\bar{M}$ maps arrows $\langle T, \mathrm{id} \rangle$ in P to $\langle T, \mathrm{id} \rangle$ in Q.*

This theorem is essentially a lax version of the equivalence between **CAT**-valued functors and split fibrations. A proof by diagram chasing can be found in the appendix. The advantage of the fibrational formulation is that it reduces the 2-categorical notion of lax transformations to the 1-categorical notion of functors. Consequently, 3-categorical concepts can be avoided when talking about transformations of modular models, such as the following concept of *liftings*.

Definition 5.5. A *lifting* $\bar{l}$ for a modular model $\bar{M}$ is a natural transformation $\bar{l} : \mathrm{Id} \to K \circ \bar{M}$ such that $P \circ \bar{l} = \iota \circ P$ where $\iota : \mathrm{Id} \to - + \tilde{\Gamma}$ is the coprojection in $\mathcal{F}$ and P, Q are as in (23):

$$\begin{tikzcd}
\mathcal{F}\text{-}\mathbf{Alg} \arrow[r, "\bar{M}"] \arrow[d, "P"'] \arrow[rr, bend left, "\mathrm{Id}"] & (\mathcal{F} + \tilde{\Gamma})\text{-}\mathbf{Alg} \arrow[r, "K"] \arrow[d, "Q"'] \arrow[dr, phantom, "\lrcorner", very near start] \arrow[u, phantom, "\Downarrow \bar{l}"] & \mathcal{F}\text{-}\mathbf{Alg} \arrow[d, "P"] \\
\mathcal{F} \arrow[r, "\mathrm{Id}"] \arrow[rr, bend right, "\mathrm{Id}"'] & \mathcal{F} \arrow[r, "(-)+\tilde{\Gamma}"] \arrow[d, phantom, "\iota \Uparrow"] & \mathcal{F} \\
 & &
\end{tikzcd}$$

Also, with the fibrational formulation, the 'dependently typed' mapping sending every $\tilde{\Sigma} \in \mathcal{F}$ to its initial algebra $\mu\hat{\Sigma}$ in $\hat{\Sigma}$-**Alg** now can be conveniently formulated as a functor $(-)^\star : \mathcal{F} \to \mathcal{F}\text{-}\mathbf{Alg}$:

$$\tilde{\Sigma} \mapsto \langle \hat{\Sigma},\ T_\Sigma,\ \mu\hat{\Sigma}, \alpha^\Sigma : \Sigma(\mu\hat{\Sigma}) \to \mu\hat{\Sigma} \rangle \quad (T : \tilde{\Sigma} \to \tilde{\Gamma}) \mapsto \langle T,\ u : \langle \mu\hat{\Sigma},\ \alpha^\Sigma \rangle \to T \langle \mu\hat{\Gamma},\ \alpha^\Gamma \rangle \rangle \tag{24}$$

where u is the unique $\hat{\Sigma}$-homomorphism out of the initial algebra $\mu\hat{\Sigma}$.

Example 5.6. For a trivial example of modular models, let $\mathcal{F}$ be the monoidal theory family containing only the theory **Mon** of monoids in $\mathscr{E}$ with the identity translation $\mathbf{Mon} \to \mathbf{Mon}$. In this case, $\mathcal{F}$-**Alg** is exactly the category $\mathbf{Mon}(\mathscr{E})$ of monoids in $\mathscr{E}$. A modular model M with a lifting l of $\langle \mathbf{Mon}, \mathrm{id} \rangle$ in $\mathcal{F}$ is precisely a (covariant) *monoid transformer* [Jaskelioff and Moggi 2010]:
$\begin{tikzcd} \mathbf{Mon}(\mathscr{E}) \arrow[r, bend left, "\mathrm{Id}"] \arrow[r, bend right, "M"'] \arrow[r, phantom, "\Downarrow l"] & \mathbf{Mon}(\mathscr{E}) \end{tikzcd}$. Thus modular models subsume monoid transformers.

Example 5.7. For a concrete example, let $\mathscr{E}$ be $\langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ for lκp $\mathscr{C}$. A strict modular model M for the theory $\mathrm{E{\scriptstyle T}}_E$ of *exception throwing* (Example 3.6) in the family $\mathbf{ALG}(\mathscr{E})$ of algebraic operations is given by a family of functors $M_{\tilde{\Sigma}} : \tilde{\Sigma}\text{-}\mathbf{Alg} \to (\tilde{\Sigma} + \mathrm{E{\scriptstyle T}}_E)\text{-}\mathbf{Alg}$ natural in $\tilde{\Sigma} \in \mathbf{ALG}(\mathscr{E})$. Recall that objects of $\tilde{\Sigma}$-**Alg** are tuples as follows satisfying certain equations:

$$\langle A \in \mathscr{E}, \quad \alpha : \Sigma \circ A \to A, \quad \eta^A : \mathrm{Id} \to A, \quad \mu^A : A \circ A \to A \rangle.$$

Each of them is mapped by $M_{\tilde{\Sigma}}$ to a $(\tilde{\Sigma} + \mathrm{ET}_E)$-algebra carried by the *exception monad transformer* $C_A = A \circ (\mathbb{E} + \mathrm{Id})$, where $\mathbb{E}$ is the E-fold product of 1 in $\mathbf{Endo}_\kappa(\mathscr{C})$. The carrier is equipped with operations $[\alpha^\sharp, \beta] : (\Sigma \circ C_A) + \mathbb{E} \longrightarrow C_A$, where

$$\alpha^\sharp = [\![s : \Sigma, a : A, e : \mathbb{E} + \mathrm{Id} \vdash (\alpha(s, a),\ e) : C_A]\!] \qquad \beta = [\![e : \mathbb{E} \vdash (\eta^A,\ \mathsf{inj}_1\, e) : C_A]\!]$$

and C_A has the following monad structure:

$$\eta^C = [\![\cdot \vdash (\eta^A, \mathsf{inj}_2(*)) : C_A]\!] \qquad \mu^C = [\![a : A,\ e : \mathbb{E} + \mathrm{Id},\ a' : A,\ e' : \mathbb{E} + \mathrm{Id} \vdash \mathsf{let}\ (a'', e'') = d(e, a')\ \mathsf{in}\ (\mu^A(a, a''), \mu^{\mathbb{E}+\mathrm{Id}}(e'', e'))]\!]$$

where $d : (\mathbb{E} + \mathrm{Id}) \circ A \longrightarrow A \circ (\mathbb{E} + \mathrm{Id})$ is a distributive law[3]:

$$e : \mathbb{E} + \mathrm{Id},\ a : A\ \vdash \mathsf{case}\ e\ \mathsf{of}\ \{\ \mathsf{inj}_1\, e' \mapsto (\eta^A, \mathsf{inj}_1 e');\ \ \mathsf{inj}_2 * \mapsto (a, \mathsf{inj}_2 *) : C_A\},$$

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

$$x : \mathbb{E} + \mathrm{Id},\ y : \mathbb{E} + \mathrm{Id}\ \vdash \mathsf{case}\ x\ \mathsf{of}\ \{\ \mathsf{inj}_1\, e \mapsto \mathsf{inj}_1\, e;\ \ \mathsf{inj}_2 * \mapsto y\} : \mathbb{E} + \mathrm{Id}.$$

The arrow mapping of $M_{\tilde{\Sigma}}$ sends a $\hat{\Sigma}$-homomorphism $h : A \longrightarrow B$ to $h \circ (\mathbb{E} + I) : C_A \longrightarrow C_B$. The modular model M has a lifting $l_{\tilde{\Sigma}, \langle A,\ \alpha \rangle} = [\![a : A \vdash (a,\ \mathsf{inj}_2\ *) : C_A]\!]$.

5.2 Interpretation with Modular Models

The point of modular models might be clearer by seeing how they are used to interpret abstract syntax. First recall that the abstract syntax of terms of some equational system $\hat{\Gamma}$ is modelled by the initial algebra $\mu\hat{\Gamma}$, then for every ordinary model $\langle A, \alpha \rangle$ of $\hat{\Gamma}$, the unique $\hat{\Gamma}$-homomorphism from $\mu\hat{\Gamma}$ to A is the *interpretation* of the syntax using the model A.

Now let M be a modular model of some theory $\tilde{\Gamma} = \langle \hat{\Gamma}, T_\Gamma \rangle$ in some monoidal theory family $\mathcal{F}$. By Definition 4.1, every $\tilde{\Sigma} = \langle \hat{\Sigma}, T_\Sigma \rangle \in \mathcal{F}$ has free algebras and thus an initial algebra $\langle \mu\hat{\Sigma}, \alpha^\Sigma \rangle$, which is mapped by $M_{\tilde{\Sigma}} : \tilde{\Sigma}\text{-}\mathbf{Alg} \longrightarrow (\tilde{\Sigma} + \tilde{\Gamma})\text{-}\mathbf{Alg}$ to an algebra of $\tilde{\Sigma} + \tilde{\Gamma}$. Then the initial algebra $\mu(\tilde{\Sigma} + \tilde{\Gamma})$ of (the equational system part of) $\tilde{\Sigma} + \tilde{\Gamma}$ induces a unique homomorphism:

$$\tilde{h}_{\tilde{\Sigma}} : \mu(\tilde{\Sigma} + \tilde{\Gamma}) \longrightarrow M_{\tilde{\Sigma}} \langle \mu\hat{\Sigma}, \alpha^\Sigma \rangle. \tag{25}$$

The intuition is that this morphism *modularly* interprets the $\hat{\Gamma}$-operations in the abstract syntax $\mu(\tilde{\Sigma} + \tilde{\Gamma})$ with a modular model M, leaving operations from the other theory $\tilde{\Sigma}$ uninterpreted. For example, with the modular model in Example 5.7, we can interpret terms of exception throwing mixed with other algebraic operations: (25) specialises to $h_{\tilde{\Sigma}} : \mu(\tilde{\Sigma} + \mathrm{ET}_E) \longrightarrow (\mu\hat{\Sigma} \circ (\mathbb{E} + \mathrm{Id}))$.

Specially, let $\tilde{\Sigma}$ be the theory $\langle \mathbf{Mon}, \mathrm{id} \rangle$ of monoids (which is always in $\mathcal{F}$ since it is the initial object of $\mathcal{F}$ and $\mathcal{F}$ is closed under finite coproducts), and then $\mu\hat{\Sigma}$ is the initial monoid I and $\mu(\tilde{\Sigma} + \tilde{\Gamma}) \cong \mu\hat{\Gamma}$. In this case, the morphism $\tilde{h}_{\tilde{\Sigma}} : \mu\hat{\Gamma} \longrightarrow \bar{M}\langle I, \alpha^I \rangle$ (25) interprets the abstract syntax $\mu\hat{\Gamma}$ without anymore uninterpreted operations.

The interpretation (25) can be formulated as a natural transformation in $\tilde{\Sigma}$ by using the functor $(-)^\star : \mathcal{F} \longrightarrow \mathcal{F}\text{-}\mathbf{Alg}$ (24) sending every $\tilde{\Sigma} \in \mathcal{F}$ to its initial algebra in $\mathcal{F}\text{-}\mathbf{Alg}$.

Proposition 5.8. *Given a modular model M of $\tilde{\Gamma} \in \mathcal{F}$, there is a natural transformation*

$$h^M : (- + \tilde{\Gamma})^\star \longrightarrow K\bar{M}(-)^\star : \mathcal{F} \longrightarrow \mathcal{F}\text{-}\mathbf{Alg} \tag{26}$$

such that all $h^M_{\tilde{\Sigma}} = \langle \mathrm{id}_{\tilde{\Sigma}+\tilde{\Gamma}}, \tilde{h}_{\tilde{\Sigma}} \rangle$, where $\bar{M} : \mathcal{F}\text{-}\mathbf{Alg} \longrightarrow (\mathcal{F} + \tilde{\Gamma})\text{-}\mathbf{Alg}$ is the equivalent form of M in Theorem 5.4, and $K : (\mathcal{F} + \tilde{\Gamma})\text{-}\mathbf{Alg} \longrightarrow \mathcal{F}\text{-}\mathbf{Alg}$ is defined as in (23).

[3]The syntax case e of $\{\mathsf{inj}_1\ x \mapsto t_1;\ \mathsf{inj}_2\ u \mapsto t_2\}$ is the eliminator of coproducts.

Remark. Since $\tilde{\Sigma} + \tilde{\Gamma}$ extends the theory of monoids, the interpretation (26) is always a monoid morphism, and thus it preserves monoid multiplication μ. This is called the *semantic substitution lemma* [Tennent 1991] for $\mathscr{E} = \langle \mathbf{Set}^{\mathbf{Fin}}, \bullet, V \rangle$ since μ stands for substitution in this case.

Liftings of modular models (Definition 5.5) are useful for *reusing existing interpretations*. Suppose that some operations are first modelled by a theory $\tilde{\Sigma}$ and interpreted with some model $A \in \tilde{\Sigma}\text{-}\mathbf{Alg}$, and later some new operations $\tilde{\Gamma}$ are added, and $\tilde{\Gamma}$ has a modular model M with a lifting l. Then by the initiality of $\mu\tilde{\Sigma}$, the following diagram commutes:

$$\begin{tikzcd}
\text{Syntax} & \mu\tilde{\Sigma} \arrow[r, "\mu\iota_1"] \arrow[d, "u_1"'] & \mu(\tilde{\Sigma} + \tilde{\Gamma}) \arrow[d, "u_2"] & \\
\text{Semantics} & A \arrow[r, "l_A"] & M_{\tilde{\Sigma}}A & \text{in } \tilde{\Sigma}\text{-}\mathbf{Alg}
\end{tikzcd}$$

where vertical arrows u_i are the unique homomorphisms out of initial algebras. This implies that interpretations of existing programs $\mu\tilde{\Sigma}$ can be reused with l, without the need to re-interpreting existing programs by $u_2 \cdot \mu\iota_1$.

6 CONSTRUCTIONS OF MODULAR MODELS

In this section we show several general constructions and concrete examples of modular models.

6.1 Modular Models from Monoid Transformers

Monad transformers, and more generally Jaskelioff and Moggi [2010]'s *monoid transformers*, map every monoid M to another TM, together with a lifting $M \to TM$. Modular models can be thought of as more elaborate versions of monoid transformers, sending *monoids with operations* to monoids with more operations, except that we do not require liftings for modular models. However, monoid transformers can sometimes be upgraded to modular models, and there are two general results: one for theories in $\mathbf{ALG}(\mathscr{E})$ from monoid transformers (Theorem 6.1) and ordinary models (Corollary 6.2), another for theories in $\mathbf{SCP}(\mathscr{E})$ from *functorial monoid transformers* (Theorem 6.4).

Theorem 6.1. *For each $\tilde{\Gamma} = \langle \hat{\Gamma}, T_\Gamma \rangle \in \mathbf{ALG}(\mathscr{E})$ over a cocordial monoidal category $\mathscr{E}$, a functor $H : \mathbf{Mon}(\mathscr{E}) \to \hat{\Gamma}\text{-}\mathbf{Alg}$ and a $\tau : \mathrm{Id} \to T_\Gamma \circ H$ can be extended to a strict modular model $\bar{M}$ of $\tilde{\Gamma}$ such that the diagram on the right below commutes, and $\bar{M}$ has a lifting $\bar{l}_{\langle \langle \hat{\Sigma}, T_\Sigma \rangle,\, A,\, \alpha \rangle} = \tau_{\langle A,\, T_\Sigma \alpha \rangle}$:*

$$\begin{tikzcd}
 & \hat{\Gamma}\text{-}\mathbf{Alg} \arrow[dr, "T_\Gamma"] & \\
\mathbf{Mon}(\mathscr{E}) \arrow[ur, "H"] \arrow[rr, "\mathrm{Id}"'] \arrow[rr, phantom, "\Uparrow \tau", shift left=3] & & \mathbf{Mon}(\mathscr{E})
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathscr{F}\text{-}\mathbf{Alg} \arrow[r, "\bar{M}"] \arrow[d] & (\mathscr{F} + \tilde{\Gamma})\text{-}\mathbf{Alg} \arrow[d] \\
\mathbf{Mon}(\mathscr{E}) \arrow[r, "H"'] & \hat{\Gamma}\text{-}\mathbf{Alg}
\end{tikzcd} \tag{27}$$

where the unlabelled vertical arrows are the evident projection functors.

Proof. Every algebraic operation $\alpha : S \,\Box\, A \to A$ and monoid morphism $\tau_A : A \to H_A$, there is always a lifting $[\![s : S, h : H_A \vdash \mu^H(\tau_A(\alpha(s, \eta^A)),\ h) : H_A]\!]$. Details can be found in the appendix. □

Example 5.7 is in fact 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 3.8) yields a modular model of St_S in $\mathbf{ALG}(\mathbf{Endo}_\kappa(\mathscr{C}))$. 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. Moreover, it allows us to obtain modular models of theories in $\mathbf{ALG}(\mathscr{E})$ from ordinary models by taking coproducts of monoids.

Corollary 6.2. Let $\hat{A} = \langle A, \alpha \rangle \in \hat{\Gamma}\text{-}\mathbf{Alg}$ be an ordinary model of $\hat{\Gamma}\text{-}\mathbf{Alg}$ for $\tilde{\Gamma} = \langle \hat{\Gamma}, T_\Gamma \rangle \in \mathbf{ALG}(\mathscr{E})$. By Fiore and Hur [2009, Theorem 6.1], the category of monoids in $\mathscr{E}$ is cocomplete for $\mathscr{E}$ cocordial.

Thus we can take coproducts of monoids, and define a functor $H : \mathbf{Mon}(\mathscr{E}) \to \hat{\Gamma}\text{-}\mathbf{Alg}$ mapping every monoid $\hat{M}$ to the $\hat{\Gamma}$-algebra $\hat{M} +_{\mathbf{Mon}} (T_\Gamma \hat{A})$ equipped with

$$g : \Gamma,\ \ m : \hat{M} +_{\mathbf{Mon}} (T_\Gamma \hat{A}) \vdash \mu(\mathsf{inj}_{\mathbf{Mon},2}\ (\alpha(g, \eta^{T_\Gamma \hat{A}})),\ m) : \hat{M} +_{\mathbf{Mon}} (T_\Gamma \hat{A})$$

Together with the coprojection $\tau_{\hat{M}} : \hat{M} \to \hat{M} +_{\mathbf{Mon}} (T_\Gamma \hat{A})$, H extends to a modular model of $\tilde{\Gamma}$.

Now we move on to the family $\mathbf{SCP}(\mathscr{E})$ of scoped operations. Let us again start with an example.

Example 6.3. The theory Ec of *exception throwing and catching* in Example 3.7 is in the family $\mathbf{SCP}(\mathscr{E})$ for $\mathscr{E} = \langle \mathbf{Endo}_\kappa(\mathbf{Set}), \circ, \mathrm{Id}\rangle$. A strict modular model for it can be constructed by extending the modular model of throwing in Example 5.7 with (i) a model for catching and (ii) a way to lift existing scoped operations on M to being on $C_A = M \circ (1 + \mathrm{Id})$.

(i) To equip the carrier $C_A = A \circ (1 + \mathrm{Id})$ with an operation $catch : (C_A \times C_A) \circ C_A \to C_A$, we denote by s the following canonical strength in $\mathbf{Endo}_\kappa(\mathbf{Set})$:

$$C_A \times C_A = (A \circ (1 + \mathrm{Id})) \times C_A \to A \circ ((1 + \mathrm{Id}) \times C_A) \cong A \circ (C_A + \mathrm{Id} \times C_A).$$

Then *catch* is $s \circ C_A$ followed by the arrow denoted by the following term:

$$\begin{aligned} a : A,\ b : (C_A + \mathrm{Id} \times C_A),\ k : C_A \vdash \mathsf{case}\ b\ \mathsf{of}\ \{\ & \mathsf{inj}_1\ x \mapsto \mu^C((a, \mathsf{inj}_2 *), \mu^C(x, k)) \\ & \mathsf{inj}_2\ y \mapsto \mathsf{let}\ * = \pi_1 y\ \mathsf{in}\ \mu^C((a, \mathsf{inj}_2 *), k)\} : C_A \end{aligned}$$

(ii) To lift an existing scoped operation $\alpha : \Sigma \circ A \circ A \to A$ on A to C_A, we define $\alpha^\sharp : S \circ C_A \circ C_A \to C_A$ by $[\![s : S,\ a : A,\ m : 1 + \mathrm{Id},\ k : C_A \vdash \mu^C(\alpha(s,\ a,\ \eta^A),\ k) : C_A]\!]$.

Similar to Theorem 6.1, modular models for theories $\hat{\Gamma} \in \mathbf{SCP}(\mathscr{E})$ can also be obtained from monoid transformers that implement operations of $\hat{\Gamma}$. The following theorem is essentially based on Jaskelioff and Moggi [2010]'s result that that scoped operations (which they call *first-order operations*) can be lifted along what they call *functorial* monoid transformers.

Theorem 6.4. *For each $\langle \hat{\Gamma}, T_\Gamma \rangle \in \mathbf{SCP}(\mathscr{E})$ over a cocordial and* closed *monoidal category $\mathscr{E}$, a functor $H : \mathbf{Mon}(\mathscr{E}) \to \hat{\Gamma}\text{-}\mathbf{Alg}$ and a natural transformation $\tau : \mathrm{Id} \to T_\Gamma \circ H$ such that there is some $F : \mathscr{E} \to \mathscr{E}$ and $\sigma : \mathrm{Id} \to F$ satisfying $\mathbf{U} \circ T_\Gamma \circ H = F \circ \mathbf{U}$ and $\tau \circ \mathbf{U} = \sigma \circ U$ can be extended to a strict modular model M of $\langle \hat{\Gamma}, T_\Gamma \rangle$ with a lifting l such that (27) commutes and $l_{\langle \langle \hat{\Sigma}, T_\Sigma \rangle,\ A,\ \alpha \rangle} = \tau_{\langle A,\ T_\Sigma \alpha \rangle}$.*

$$\begin{tikzcd}
 & \hat{\Gamma}\text{-}\mathbf{Alg} \arrow[dr, "T_\Gamma"] & \\
\mathbf{Mon}(\mathscr{E}) \arrow[ur, "H"] \arrow[rr, "\mathrm{Id}" description] \arrow[d, "\mathbf{U}"'] & \Uparrow \tau & \mathbf{Mon}(\mathscr{E}) \arrow[d, "\mathbf{U}"] \\
\mathscr{E} \arrow[rr, bend left, "F" description] \arrow[rr, "\mathrm{Id}"'] & \Uparrow \sigma & \mathscr{E}
\end{tikzcd}$$

In comparison to Theorem 6.1, the theorem above requires a closed $\mathscr{E}$ and the monoid transformer $\langle T_\Gamma \circ H, \tau \rangle$ to be *functorial*, i.e. an extension of an endofunctor transformer $\langle F, \sigma \rangle$. This is because the retract $\epsilon : A/A \to A$ of the Cayley embedding (Example 2.2) used essentially in the proof is *not* a monoid morphism. Many monoid transformers are functorial, including the exception monad transformer underlying Example 6.3, the state monad transformer $A \mapsto (A(S \times -))^S$, and the free monad transformer (also known as the resumption monad transformer) $A \mapsto \mu X.\, A(- + FX)$ for accessible endofunctors $F : \mathscr{C} \to \mathscr{C}$ [Cenciarelli and Moggi 1993].

6.2 Free Modular Models

In this subsection we show a very general construction, the *free modular models*. The idea is to construct a modular model for an arbitrary theory $\tilde{\Gamma}$ in the theory family $\mathbf{Mon/Eqs}_c(\mathscr{E})$ by using the relative free algebra functor $\mathbf{F}_{\tilde{\Sigma}} : \tilde{\Sigma}\text{-}\mathbf{Alg} \to (\tilde{\Sigma} + \tilde{\Gamma})\text{-}\mathbf{Alg}$ from Theorem 3.12.

However, a problem is that this family of functors is *not* natural in $\tilde{\Sigma}$. To see this, consider $\mathscr{E} = \langle \mathbf{Set}, \times, 1 \rangle$. We have the following theories in the family $\mathbf{Mon/Eqs}_c(\mathscr{E})$: $\mathbf{Mon}$ with the identity translation; **Grp** with the inclusion translation $T : \mathbf{Mon} \to \mathbf{Grp}$ (Example 3.10); 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:

$$\begin{tikzcd}
\mathbf{Mon}\text{-}\mathbf{Alg} \arrow[d, "\mathbf{F}_{\mathbf{Mon}}"'] & \mathbf{Grp}\text{-}\mathbf{Alg} \arrow[l, "T"'] \arrow[d, "\mathbf{F}_{\mathbf{Grp}}"] \\
(\mathbf{Mon} + \mathbf{BLat})\text{-}\mathbf{Alg} & (\mathbf{Grp} + \mathbf{BLat})\text{-}\mathbf{Alg} \arrow[l, "T+\mathbf{BLat}"']
\end{tikzcd}$$

Given a group G, $\mathbf{F}_{\mathbf{Grp}} G$ is a certain quotient of the set inductively generated from G-elements, operations of bounded lattices, and operations of groups. The translation $T + \mathbf{BLat}$ on $\mathbf{F}_{\mathbf{Grp}} G$ simply forgets the operation of group inverse, *leaving the carrier unchanged.* On the other hand, $\mathbf{F}_{\mathbf{Mon}}$ on TG is some quotient of the set generated from G-elements, operations of bounded lattices, and operations of monoids. The carrier sets of $\mathbf{F}_{\mathbf{Mon}} TG$ and $(T + \mathbf{BLat})\mathbf{F}_{\mathbf{Grp}}$ are different: the inductively generated group inverses are still in the carrier of the latter. However, the diagram above can be made an oplax transformation, resulting in a non-strict modular model of **BLat**.

Theorem 6.5. *Let $\mathscr{E}$ be a cocordial category. For any $\tilde{\Gamma}$ in the family $\mathbf{Mon}/\mathbf{Eqs}_c(\mathscr{E})$, the family of functors $\mathbf{F}_{\tilde{\Sigma}} : \tilde{\Sigma}\text{-}\mathbf{Alg} \to (\tilde{\Sigma} + \tilde{\Gamma})\text{-}\mathbf{Alg}$ can be extended to a (non-strict) modular model.*

Proof sketch. For every $T : \tilde{\Sigma} \to \tilde{\Sigma}'$ and $A \in \tilde{\Sigma}'\text{-}\mathbf{Alg}$, there is the unit $\tilde{\Sigma}'$-homomorphism $\eta : A \to \mathbf{U}_{\tilde{\Sigma}'}\mathbf{F}_{\tilde{\Sigma}'}A$. This is mapped by T to a $\tilde{\Sigma}$-homomorphism $TA \to T\mathbf{U}_{\tilde{\Sigma}'}\mathbf{F}_{\tilde{\Sigma}'}A = \mathbf{U}_{\tilde{\Sigma}}(T + \tilde{\Gamma})\mathbf{F}_{\tilde{\Sigma}'}A$.

$$\begin{tikzcd}
\tilde{\Sigma}\text{-}\mathbf{Alg} \arrow[d, shift left, "\mathbf{F}_{\tilde{\Sigma}}"] & \tilde{\Sigma}'\text{-}\mathbf{Alg} \arrow[l, "T"'] \arrow[d, shift left, "\mathbf{F}_{\tilde{\Sigma}'}"] \\
(\tilde{\Sigma} + \tilde{\Gamma})\text{-}\mathbf{Alg} \arrow[u, shift left, "\mathbf{U}_{\tilde{\Sigma}}"] & (\tilde{\Sigma} + \tilde{\Gamma})\text{-}\mathbf{Alg} \arrow[l, "T+\tilde{\Gamma}"] \arrow[u, shift left, "\mathbf{U}_{\tilde{\Sigma}'}"]
\end{tikzcd}$$

By the universal property of $\mathbf{F}_{\tilde{\Sigma}} TA$, there is a $(\tilde{\Sigma} + \tilde{\Gamma})$-homomorphism $\mathbf{F}_{\tilde{\Sigma}}(TA) \to (T + \tilde{\Gamma})(\mathbf{F}_{\tilde{\Sigma}'}A)$, which is natural in A. This makes $\mathbf{F}$ a modular model of $\tilde{\Gamma}$. □

Example 6.6. The free modular model is remarkably general as it works for $\mathbf{Mon}/\mathbf{Eqs}_c(\mathscr{E})$. For an example, let us consider how the free modular model adds new operations to a model of λ-calculus. Let $\mathscr{E}$ be $\langle \mathbf{Set}^{\mathbf{Fin}}, \bullet, V \rangle$ (5). As we mentioned earlier (22), the syntax of λ-calculus can be presented as an equational system $\Lambda = \Sigma_{\langle O,a \rangle}\text{-}\mathbf{Mon}$ for $O = \{ \mathit{app}, \mathit{abs} \}$ with $a(\mathit{abs}) = \langle 1 \rangle$ and $a(\mathit{app}) = \langle 0, 0 \rangle$. Models of Λ can be obtained from reflexive objects $U \cong U^U$ in any cartesian closed category C: every U induces a functor $\bar{U} : \mathbf{Fin} \to \mathbf{Set}$ 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]:

$$\begin{aligned}
\mathit{abs}_U &: \bar{U}^V n \cong \mathscr{C}(U^{n+1}, U) \cong \mathscr{C}(U^n, U^U) \cong \mathscr{C}(U^n, U) \cong \bar{U} n \\
\mathit{app}_U &: (\bar{U} \times \bar{U}) n \cong \mathscr{C}(U^n, U \times U) \cong \mathscr{C}(U^n, U^U \times U) \to \mathscr{C}(U^n, U) \cong \bar{U} n
\end{aligned}$$

Now consider the theory St_S of mutable state (Example 3.8) for some finite set S. Its free modular model maps the Λ-model on $\bar{U}$ to a $(\Lambda +_{\mathbf{Mon}} \mathrm{St}_S)$-model whose carrier is the initial algebra

$$\mu X.\, \bar{U} + X \bullet X + V + X^V + X \times X + \textstyle\prod_S X + \coprod_S X : \mathbf{Fin} \to \mathbf{Set}$$

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 $[\eta_U, \mu_U, \mathit{abs}_U, \mathit{app}_U]$ of $\bar{U}$.

6.3 Composition and Fusion of Modular Models

Modular models of different theories are *composable*, justifying the name 'modular models'. Composites of modular models admit a *fusion lemma*, saying that interpreting a term with two modular models sequentially is equal to interpreting with the composite of the two models.

Definition 6.7. Given a (resp. strict) modular model $\bar{M}$ of $\tilde{\Gamma} \in \mathcal{F}$ and another $\bar{N}$ of $\tilde{\Phi} \in \mathcal{F}$ in the form of Theorem 5.4, their *composite* $\bar{M} \triangleright \bar{N}$ is a (resp. strict) modular model of $\tilde{\Gamma} + \tilde{\Phi}$:

$$\bar{M} \triangleright \bar{N} : \mathcal{F}\text{-}\mathbf{Alg} \to (\mathcal{F} + \tilde{\Gamma} + \tilde{\Phi})\text{-}\mathbf{Alg}$$

defined by the pullback property (23) for $(\mathcal{F} + \tilde{\Gamma} + \tilde{\Phi})$-**Alg** and the commutativity of the diagram:

$$\begin{tikzcd}
\mathcal{F}\text{-}\mathbf{Alg} \arrow[r, "\bar{M}"] \arrow[d, "P"'] & (\mathcal{F} + \tilde{\Gamma})\text{-}\mathbf{Alg} \arrow[r, "K"] \arrow[d, "Q"'] & \mathcal{F}\text{-}\mathbf{Alg} \arrow[r, "\bar{N}"] \arrow[d, "P"'] & (\mathcal{F} + \tilde{\Phi})\text{-}\mathbf{Alg} \arrow[r, "K"] \arrow[d, "Q'"'] & \mathcal{F}\text{-}\mathbf{Alg} \arrow[d, "P"'] \\
\mathcal{F} \arrow[r, "\mathrm{Id}"'] & \mathcal{F} \arrow[r, "-+\tilde{\Gamma}"'] & \mathcal{F} \arrow[r, "\mathrm{Id}"'] & \mathcal{F} \arrow[r, "-+\tilde{\Phi}"'] & \mathcal{F}
\end{tikzcd}$$

Also, liftings $\bar{l}^N$ of N and $\bar{l}^M$ of M in the form of Theorem 5.4 compose a lifting $\bar{l}^N \circ \bar{l}^M$ of $\bar{M} \triangleright \bar{N}$.

Example 6.8. Let M_E be the modular model of *exception* throwing and catching (Example 6.3), and M_S be the modular model of *mutable state* arising from the state monad transformer [Liang et al. 1995] by Theorem 6.1. The composite $M_E \triangleright M_S$ is a modular model of the coproduct $\mathrm{Ec} + \mathrm{St}_S$ of the theories of exception and mutable state.

Remark. Although the coproduct of theories is commutative, $\hat{\Gamma} + \hat{\Phi} \cong \hat{\Phi} + \hat{\Gamma}$, the composition of modular models is *not*. For example, the opposite order $M_S \triangleright M_E$ of composing the modular models in Example 6.8 gives rise to a different modular model of the coproduct $\mathrm{Ec} + \mathrm{St}_S$, in which state and exception *interact* differently. Informally, when an exception is caught, $M_E \triangleright M_S$ rolls back to the state before the *catch*, whereas $M_S \triangleright M_E$ keeps the state. Both behaviours are desirable depending on the application, so the language designer needs to determine the correct order of composing modular models according to the desired interaction.

A composite model $N \triangleright M$ can be used for interpreting $(\tilde{\Sigma} + (\tilde{\Gamma} + \tilde{\Phi}))^\star$ by (26). Since $(\tilde{\Sigma} + (\tilde{\Gamma} + \tilde{\Phi}))^\star \cong ((\tilde{\Sigma} + \tilde{\Gamma}) + \tilde{\Phi})^\star$, it can also be interpreted by using N and M sequentially. The results of these two ways can be shown to be equal by the initiality of $(\tilde{\Sigma} + (\tilde{\Gamma} + \tilde{\Phi}))^\star$.

Lemma 6.9 (Fusion). *Given modular models M of $\tilde{\Gamma}$ and N of $\tilde{\Phi}$, the following diagram commutes:*

$$\begin{tikzcd}
(\tilde{\Sigma} + (\tilde{\Gamma} + \tilde{\Phi}))^\star \arrow[r, "\cong"] \arrow[d, "h^{M \triangleright N}_{\tilde{\Sigma}}"'] & ((\tilde{\Sigma} + \tilde{\Gamma}) + \tilde{\Phi})^\star \arrow[d, "h^N_{\tilde{\Sigma}+\tilde{\Gamma}}"] \\
K\bar{N}K\bar{M}\tilde{\Sigma}^\star & K\bar{N}(\tilde{\Sigma} + \tilde{\Gamma})^\star \arrow[l, "K\bar{N}h^M_{\tilde{\Sigma}}"']
\end{tikzcd}$$

This result is an instance of *short-cut fusion* [Ghani and Johann 2007; Gill et al. 1993]. It combines *two* consecutive interpretations of terms $(\hat{\Sigma} + \hat{\Gamma} + \hat{\Phi})^\star$ into *one*, eliminating the intermediate result $(\hat{\Sigma} + \hat{\Gamma})^\star$. Thus it is useful for optimising the performance and reasoning about their interactions.

6.4 Modular Models in Symmetric Monoidal Categories

Finally, we show some constructions of modular models that are only possible in symmetric monoidal categories $\mathscr{E}$, such as $\langle \mathscr{C}, \times, 1 \rangle$ for cartesian monoids and $\langle \mathbf{Endo}_\kappa(\mathbf{Set}), \star, \mathrm{Id} \rangle$ for applicatives.

In a symmetric $\mathscr{E}$, any two monoids $\langle A, \mu^A, \eta^A \rangle$ and $\langle B, \mu^B, \eta^B \rangle$ compose to a monoid $A \mathbin{\square} B$:

$$\mu^{A \square B} = \left((A \mathbin{\square} B) \mathbin{\square} (A \mathbin{\square} B) \cong (A \mathbin{\square} A) \mathbin{\square} (B \mathbin{\square} B) \xrightarrow{\mu^A \square \mu^B} A \mathbin{\square} B\right) \qquad \eta^{A \square B} = \eta^A \mathbin{\square} \eta^B$$

Moreover, there is a canonical way to lift scoped operations $\alpha : \Gamma \mathbin{\square} A \mathbin{\square} A \to A$ on A to $A \mathbin{\square} B$:

$$\Gamma \mathbin{\square} (A \mathbin{\square} B) \mathbin{\square} (A \mathbin{\square} B) \cong \Gamma \mathbin{\square} (A \mathbin{\square} A) \mathbin{\square} (B \mathbin{\square} B) \xrightarrow{\alpha \square \mu^B} A \mathbin{\square} B$$

Since algebraic operations are special cases of scoped operations, they can be lifted similarly. This allows us to upgrade ordinary models of theories in $\mathbf{ALG}(\mathscr{E})$ or $\mathbf{SCP}(\mathscr{E})$ to modular models.

Theorem 6.10 (Independent Combination). *Let $\mathscr{E}$ be a symmetric cocordial category and $\mathcal{F}$ be* $\mathbf{ALG}(\mathscr{E})$ *or* $\mathbf{SCP}(\mathscr{E})$*. For each $\tilde{\Gamma} \in \mathcal{F}$, every ordinary model $\hat{A} \in \tilde{\Gamma}$-*$\mathbf{Alg}$ *induces a strict modular model M of $\tilde{\Gamma}$ such that $M_{\tilde{\Sigma}}\langle B, \beta\rangle$ is carried by $A \,\Box\, B$, and M has a lifting $l_{\tilde{\Sigma},\langle B,\beta\rangle} = [\![b : B \vdash (\eta^A,\ b) : A \,\Box\, B]\!]$.*

For $\mathscr{E} = \langle \mathbf{Endo}_\kappa(\mathbf{Set}), \star, \mathrm{Id}\rangle$, the intuition for this construction $A \star B$ is that two kinds of applicative-computations A and B are combined in the way that they execute *independently*, and operations act on $A \star B$ 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.

Theorem 6.11 (Dependent Combination). *Let $\mathscr{E}$ be a $\langle \mathbf{Endo}_\kappa(\mathbf{Set}), \star, \mathrm{Id}\rangle$ and $\mathcal{F}$ be* $\mathbf{ALG}(\mathscr{E})$ *or* $\mathbf{SCP}(\mathscr{E})$*. For each $\tilde{\Gamma} \in \mathcal{F}$, every ordinary model $\hat{A} \in \tilde{\Gamma}$-*$\mathbf{Alg}$ *induces a strict modular model M of $\tilde{\Gamma}$ such that $M_{\tilde{\Sigma}}\langle B, \beta\rangle$ is carried by $A \circ B$, and M has a lifting $l_{\tilde{\Sigma},\langle B,\beta\rangle} = \eta^A \circ B$.*

Example 6.12. To highlight the difference between Theorem 6.10 and Theorem 6.11, let $\hat{A}$ be the applicative functor induced by the exception monad $\mathbb{E} + \mathrm{Id}$. It is a model of the applicative version of the theory $\textsc{Et}_E$ of exception *throwing*, equipped with an operation $\mathit{throw} : \mathbb{E} \star (\mathbb{E} + \mathrm{Id}) \longrightarrow (\mathbb{E} + \mathrm{Id})$. Using Theorem 6.11, it can be extended to a modular model using $(\mathbb{E} + \mathrm{Id}) \circ B \cong (\mathbb{E} + B)$ for all applicatives B. In this model, it holds that for all elements $x, y \in (\mathbb{E} + B)X$, $\mathit{throw}\ \langle e, x\rangle = \mathit{inj}_1\ e = \mathit{throw}\ \langle e, y\rangle$, which means that exception throwing discards any B-computation. But it is not true for the independent composition $(\mathbb{E} + \mathrm{Id}) \star B$ using Theorem 6.10.

Phasing. As our final example, we show an interesting modular model for *multi-phase computation*, generalising the construction that Kidney and Wu [2021] introduce for *breadth-first search*. The theory $\textsc{Pha} \in \mathbf{SCP}(\mathscr{E})$ has a unary scoped operation $\mathit{later} : A \,\Box\, A \cong I \,\Box\, A \,\Box\, A \longrightarrow A$. The intuition is that a program may have multiple phases of execution, and the operation $\mathit{later}\ \langle p, k\rangle$ delays the execution of p by a phase, and continues as k. For example, the program

$$\mu(\mathit{later}\ \langle \mathit{later}\langle p_3, p_{21}\rangle,\ p_{11}\rangle,\ \mathit{later}\ \langle p_{22}, p_{12}\rangle)$$

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.

Given a monoid $\hat{A} = \langle A, \mu^A, \eta^A\rangle$ in a symmetric closed cocordial monoidal category, Kidney and Wu [2021]'s idea can be abstracted as equipping the free monoid $S_A = \mu X.\ A \,\Box\, X + I$ over A with a nonstandard monoid structure $\langle S_A, \mu^{S_A}, \eta^{S_A}\rangle$ with $\eta^{S_A} = [\![\cdot \vdash \mathit{out}^\circ\ (\mathsf{inj}_2 *)]\!]$ and μ^{S_A} being

$$\begin{aligned} s : S_A, t : S_A \vdash \mathsf{case}\ (\mathit{out}\ s, \mathit{out}\ t)\ \mathsf{of}\ \{&(\mathsf{inj}_1\ (a, x),\ \mathsf{inj}_1\ (a', y)) \mapsto \mathit{out}^\circ(\mathsf{inj}_1\ (\mu^A(a, a'), \mu^{S_A}(x, y)); \\ &(\mathsf{inj}_2 *,\ y) \mapsto \mathit{out}^\circ\ y; \quad (x,\ \mathsf{inj}_2 *) \mapsto \mathit{out}^\circ\ x\} \end{aligned}$$

where $\mathit{out} : (S_A \cong A \,\Box\, S_A + I) : \mathit{out}^\circ$ is the isomorphism for the initial algebra. Note that the use of variable x and a' does not match their order in the context, so we need a symmetric monoidal category, and we also need closedness for implementing structural recursion on the initial algebra S_A. The intuition is that $S_A = \mu X. A \,\Box\, X + I$ 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 $[\![(p : S_A, k : S_A \vdash \mu^{S_A}(\mathit{out}^\circ(\mathsf{inj}_1(\eta^A, p), k)) : S_A]\!]$.

Proposition 6.13. *Let $\mathscr{E}$ be a symmetric closed cocordial monoidal category. There is a modular model M of the theory $\textsc{Pha}$ of phasing in* $\mathbf{SCP}(\mathscr{E})$*, such that $M_{\tilde{\Sigma}}\langle A, \alpha\rangle$ is carried by $\mu X.\ A \,\Box\, X + I$.*

Concluding Remark. Inspired by effect handlers and monad transformers, we have developed a framework of modular models of monoids equipped with operations. As future work, we wish to explore the connections of our framework to Lawvere-style algebraic theories, and consider variations of modular models that are not covariant in their domains, which will encompass modular models based on the continuation monad transformer. Another important direction is the design of a syntactic calculus for monoidal theory families and modular models.

REFERENCES

J. Adamek and J. Rosicky. 1994. *Locally Presentable and Accessible Categories.* Cambridge University Press. https://doi.org/10.1017/CBO9780511600579

Jiří Adámek. 1974. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae* 015, 4 (1974), 589–602. http://eudml.org/doc/16649

Casper Bach Poulsen and Cas van der Rest. 2023. Hefty Algebras: Modular Elaboration of Higher-Order Algebraic Effects. *Proc. ACM Program. Lang.* 7, POPL, Article 62 (jan 2023), 31 pages. https://doi.org/10.1145/3571255

Pietro Cenciarelli and Eugenio Moggi. 1993. *A Syntactic Approach to Modularity in Denotational Semantics.* Technical Report. In Proceedings of the Conference on Category Theory and Computer Science. https://doi.org/10.1.1.41.7807

P. M. Cohn. 1981. *Universal Algebra.* Springer Dordrecht. https://doi.org/10.1007/978-94-009-8399-1

Brian Day. 1970. On closed categories of functors. In *Reports of the Midwest Category Seminar IV*, S. MacLane, H. Applegate, M. Barr, B. Day, E. Dubuc, Phreilambud, A. Pultr, R. Street, M. Tierney, and S. Swierczkowski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–38.

Andrzej Filinski. 1994. Representing Monads. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Portland, Oregon, USA) *(POPL '94)*. Association for Computing Machinery, New York, NY, USA, 446–457. https://doi.org/10.1145/174675.178047

Andrzej Filinski. 1999. Representing Layered Monads. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Antonio, Texas, USA) *(POPL '99)*. Association for Computing Machinery, New York, NY, USA, 175–188. https://doi.org/10.1145/292540.292557

Marcelo Fiore. 2008. Second-Order and Dependently-Sorted Abstract Syntax. In *Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science (LICS '08)*. IEEE Computer Society, USA, 57–68. https://doi.org/10.1109/LICS.2008.38

Marcelo Fiore and Chung-Kil Hur. 2007. Equational Systems and Free Constructions. In *Proceedings of the 34th International Conference on Automata, Languages and Programming* (Wrocław, Poland) *(ICALP'07)*. Springer-Verlag, Berlin, Heidelberg, 607–618. https://doi.org/10.5555/2394539.2394612

Marcelo Fiore and Chung-Kil Hur. 2009. On the construction of free algebras for equational systems. *Theoretical Computer Science* 410, 18 (2009), 1704–1729. https://doi.org/10.1016/j.tcs.2008.12.052 Automata, Languages and Programming (ICALP 2007).

Marcelo Fiore and Ola Mahmoud. 2014. Functorial Semantics of Second-Order Algebraic Theories. https://doi.org/10.48550/ARXIV.1401.4697

Marcelo Fiore and Philip Saville. 2017. List Objects with Algebraic Structure. In *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017) (Leibniz International Proceedings in Informatics (LIPIcs))*, Dale Miller (Ed.), Vol. 84. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 16:1–16:18. https://doi.org/10.4230/LIPIcs.FSCD.2017.16

Marcelo Fiore and Dmitrij Szamozvancev. 2022. Formal Metatheory of Second-Order Abstract Syntax. *Proc. ACM Program. Lang.* 6, POPL, Article 53 (jan 2022), 29 pages. https://doi.org/10.1145/3498715

Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. 1999. Abstract Syntax and Variable Binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999.*

Neil Ghani and Patricia Johann. 2007. Monadic augment and generalised short cut fusion. *Journal of Functional Programming* 17, 6 (2007), 731–776. https://doi.org/10.1017/S0956796807006314

Neil Ghani and Tarmo Uustalu. 2004. Coproducts of Ideal Monads. *RAIRO - Theoretical Informatics and Applications* 38, 4 (oct 2004), 321–342. https://doi.org/10.1051/ita:2004016

Neil Ghani, Tarmo Uustalu, and Makoto Hamana. 2006. Explicit substitutions and higher-order syntax. *High. Order Symb. Comput.* (2006).

Andy Gill and Edward Kmett. 2012. `mtl`: Monad classes, using functional dependencies. https://hackage.haskell.org/package/mtl.

Andrew Gill, John Launchbury, and Simon L. Peyton Jones. 1993. A Short Cut to Deforestation. In *Proceedings of the Conference on Functional Programming Languages and Computer Architecture* (Copenhagen, Denmark) *(FPCA '93)*. Association for Computing Machinery, New York, NY, USA, 223–232. https://doi.org/10.1145/165180.165214

Ralf Hinze. 2012. Kan Extensions for Program Optimisation Or: Art and Dan Explain an Old Trick. In *Mathematics of Program Construction*, Jeremy Gibbons and Pablo Nogueira (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 324–362. https://doi.org/978-3-642-31113-0_16

John Hughes. 1986. A Novel Representation of Lists and its Application to the Function "reverse". *Inf. Process. Lett.* 22 (01 1986), 141–144.

Martin Hyland. 2017. Classical lambda calculus in modern dress. *Mathematical Structures in Computer Science* 27, 5 (2017), 762–781. https://doi.org/10.1017/S0960129515000377

B. Jacobs. 1999. *Categorical Logic and Type Theory.* Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam.

Mauro Jaskelioff and Eugenio Moggi. 2010. Monad transformers as monoid transformers. *Theoretical Computer Science* 411 (12 2010), 4441–4466. https://doi.org/10.1016/j.tcs.2010.09.011

Shin-ya Katsumata. 2014. Parametric Effect Monads and Semantics of Effect Systems. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '14)*. Association for Computing Machinery, New York, NY, USA, 633–645. https://doi.org/10.1145/2535838.2535846

Shin-ya Katsumata, Dylan McDermott, Tarmo Uustalu, and Nicolas Wu. 2022. Flexible Presentations of Graded Monads. *Proc. ACM Program. Lang.* 6, ICFP, Article 123 (aug 2022), 29 pages. https://doi.org/10.1145/3547654

G.M. Kelly and A.J. Power. 1993. Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. *Journal of Pure and Applied Algebra* 89, 1 (1993), 163–179. https://doi.org/10.1016/0022-4049(93)90092-8

Donnacha Oisín Kidney and Nicolas Wu. 2021. Algebras for Weighted Search. *Proc. ACM Program. Lang.* 5, ICFP, Article 72 (aug 2021), 30 pages. https://doi.org/10.1145/3473577

Oleg Kiselyov and Hiromi Ishii. 2015. Freer Monads, More Extensible Effects. *SIGPLAN Not.* 50, 12 (aug 2015), 94–105. https://doi.org/10.1145/2887747.2804319

Satoshi Kura. 2020. Graded Algebraic Theories. In *Foundations of Software Science and Computation Structures*, Jean Goubault-Larrecq and Barbara König (Eds.). Springer International Publishing, Cham, 401–421. https://doi.org/10.1007/978-3-030-45231-5_21

Joachim Lambek. 1958. The Mathematics of Sentence Structure. *The American Mathematical Monthly* 65, 3 (1958), 154–170. http://www.jstor.org/stable/2310058

F. William Lawvere. 1963. FUNCTORIAL SEMANTICS OF ALGEBRAIC THEORIES. *Proceedings of the National Academy of Sciences* 50, 5 (1963), 869–872. https://doi.org/10.1073/pnas.50.5.869

Sheng Liang, Paul Hudak, and Mark Jones. 1995. Monad Transformers and Modular Interpreters. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, USA) *(POPL '95)*. ACM, 333–343. https://doi.org/10.1145/199448.199528

F. E. J. Linton. 1966. Some Aspects of Equational Categories. In *Proceedings of the Conference on Categorical Algebra*, S. Eilenberg, D. K. Harrison, S. MacLane, and H. Röhrl (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 84–94. https://doi.org/10.1007/978-3-642-99902-4_3

Saunders Mac Lane. 1998. *Categories for the Working Mathematician, 2nd edn.* Springer, Berlin.

Conor Mcbride and Ross Paterson. 2008. Applicative Programming with Effects. *J. Funct. Program.* 18, 1 (Jan. 2008), 1–13. https://doi.org/10.1017/S0956796807006326

Dylan McDermott and Tarmo Uustalu. 2022. Flexibly Graded Monads and Graded Algebras. In *Mathematics of Program Construction*, Ekaterina Komendantskaya (Ed.). Springer International Publishing, Cham, 102–128. https://doi.org/10.1007/978-3-031-16912-0_4

Eugenio Moggi. 1989a. *An Abstract View of Programming Languages.* Technical Report ECS-LFCS-90-113. Edinburgh University, Department of Computer Science.

E. Moggi. 1989b. Computational lambda-calculus and monads. In *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*. 14–23. https://doi.org/10.1109/LICS.1989.39155

Eugenio Moggi. 1991. Notions of computation and monads. *Information and Computation* 93, 1 (1991), 55 – 92. https://doi.org/10.1016/0890-5401(91)90052-4 Selections from 1989 IEEE Symposium on Logic in Computer Science.

Andrey Mokhov, Neil Mitchell, and Simon Peyton Jones. 2018. Build Systems à La Carte. *Proc. ACM Program. Lang.* 2, ICFP, Article 79 (jul 2018), 29 pages. https://doi.org/10.1145/3236774

Ross Paterson. 2012. Constructing applicative functors. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7342 LNCS (2012), 300–323. https://doi.org/10.1007/978-3-642-31113-0_15

Maciej Piróg, Tom Schrijvers, Nicolas Wu, and Mauro Jaskelioff. 2018. Syntax and Semantics for Operations with Scopes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, Anuj Dawar and Erich Grädel (Eds.).

G Plotkin and John Power. 2001. Semantics for Algebraic Operations. *Electronic Notes in Theoretical Computer Science* 45 (2001), 332–345. https://doi.org/10.1016/S1571-0661(04)80970-8

Gordon Plotkin and John Power. 2002. Notions of Computation Determine Monads. In *Foundations of Software Science and Computation Structures, 5th International Conference (FOSSACS 2002)*, Mogens Nielsen and Uffe Engberg (Eds.). Springer, 342–356. https://doi.org/10.1007/3-540-45931-6_24

Gordon Plotkin and John Power. 2003. Algebraic Operations and Generic Effects. *Applied Categorical Structures* 11, 1 (Feb. 2003), 69–94. https://doi.org/10.1023/A:1023064908962

Gordon Plotkin and John Power. 2004. Computational Effects and Operations: An Overview. *Electr. Notes Theor. Comput. Sci.* 73 (10 2004), 149–163. https://doi.org/10.1016/j.entcs.2004.08.008

Gordon Plotkin and Matija Pretnar. 2013. Handling Algebraic Effects. *Logical Methods in Computer Science* 9, 4 (Dec 2013). https://doi.org/10.2168/lmcs-9(4:23)2013

Jeff Polakow and Frank Pfenning. 1999. Natural Deduction for Intuitionistic Non-communicative Linear Logic. In *Proceedings of the Fourth International Conference on Typed Lambda Calculi and Applications (Lecture Notes in Computer Science)*, Vol. 1581. Springer-Verlag, 295–309. https://doi.org/10.1007/3-540-48959-2_21

John Power. 1999. Enriched lawvere theories. *Theory and Applications of Categories* 6, 7 (1999), 83–93.

John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism. In *IFIP Congress*.

Exequiel Rivas and Mauro Jaskelioff. 2017. Notions of computation as monoids. *Journal of Functional Programming* 27, September (oct 2017), e21. https://doi.org/10.1017/S0956796817000132 arXiv:1406.4823

Ian Stark. 2008. Free-algebra models for the π-calculus. *Theoretical Computer Science* 390, 2 (2008), 248–270. https://doi.org/10.1016/j.tcs.2007.09.024 Foundations of Software Science and Computational Structures.

Wouter Swierstra. 2008. Data types à la carte. *J. Funct. Program.* 18, 4 (2008), 423–436. https://doi.org/10.1017/S0956796808006758

Robert D Tennent. 1991. *Semantics of programming languages*. Vol. 1. Prentice Hall New York.

Janis Voigtländer. 2008. Asymptotic Improvement of Computations over Free Monads. In *Mathematics of Program Construction*, Philippe Audebaud and Christine Paulin-Mohring (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 388–403. https://doi.org/10.1007/978-3-540-70594-9_20

Philip Wadler. 1995. Monads for Functional Programming. In *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques-Tutorial Text*. Springer-Verlag, Berlin, Heidelberg, 24–52. https://doi.org/10.5555/647698.734146

Nicolas Wu, Tom Schrijvers, and Ralf Hinze. 2014. Effect handlers in scope. *Proceedings of the 2014 ACM SIGPLAN symposium on Haskell - Haskell '14* (2014), 1–12. https://doi.org/10.1145/2633357.2633358

Zhixuan Yang, Marco Paviotti, Nicolas Wu, Birthe van den Berg, and Tom Schrijvers. 2022. Structured Handling of Scoped Effects. Springer-Verlag, Berlin, Heidelberg, 462–491. https://doi.org/10.1007/978-3-030-99336-8_17

A SOME CALCULATIONS ABOUT GRADED MONADS

To see the correspondence between (7) and monoids $\langle M, \eta : I \longrightarrow M, \mu : M \star M \longrightarrow M \rangle$ of the monoidal product (8), we reason using some end calculus:

$$\begin{aligned}
& \mathbf{hom}(I, M) && \{\text{ We write } [A,B] \text{ for sets of functions } B^A\ \} \\
\cong\ & \textstyle\int_{c\in\mathbb{G}} \int_{x\in\mathbf{Fin}} [\coprod_{\mathbb{G}(1,c)} x,\ (Mc)x] \\
\cong\ & \textstyle\int_{c\in\mathbb{G}} \int_{x\in\mathbf{Fin}} [\mathbb{G}(1,c) \times x,\ (Mc)x] \\
\cong\ & \textstyle\int_{c\in\mathbb{G}} \int_{x\in\mathbf{Fin}} [\mathbb{G}(1,c), [x,\ (Mc)x]] \\
\cong\ & \textstyle\int_{c\in\mathbb{G}} [\mathbb{G}(1,c), (\int_{x\in\mathbf{Fin}} [x,\ (Mc)x])] && \{\text{ Yoneda lemma }\} \\
\cong\ & \textstyle\int_{x\in\mathbf{Fin}} [x,\ (M1)x] \cong \mathbf{hom}(\mathrm{Id}, M1)
\end{aligned}$$

and for the correspondence of μ:

$$\begin{aligned}
& \mathbf{hom}(M \star M, M) \\
\cong\ & \textstyle\int_{c\in\mathbb{G}} \mathbf{hom}((M \star M)c, Mc) \\
\cong\ & \textstyle\int_{c\in\mathbb{G}} \int_{x\in\mathbf{Fin}} [\int^{a,b\in\mathbb{G}} \coprod_{\mathbb{G}(a\cdot b,-)} (Ma(Mbx)), (Mc)x] \\
\cong\ & \textstyle\int_{c\in\mathbb{G}} \int_{x\in\mathbf{Fin}} \int_{a,b\in\mathbb{G}} [\mathbb{G}(a\cdot b,-), [(Ma(Mbx)), (Mc)x]] \\
\cong\ & \textstyle\int_{x\in\mathbf{Fin}} \int_{a,b\in\mathbb{G}} \int_{c\in\mathbb{G}} [\mathbb{G}(a\cdot b,-), [(Ma(Mbx)), (Mc)x]] \\
\cong\ & \textstyle\int_{x\in\mathbf{Fin}} \int_{a,b\in\mathbb{G}} [(Ma(Mbx)), (M(a\cdot b))x] \\
\cong\ & \textstyle\int_{a,b\in\mathbb{G}} \mathrm{hom}(Ma \circ Mb, M(a\cdot b))
\end{aligned}$$

This monoidal product is right closed with the right adjoint given by

$$(G \mathrel{-\!\!\star} F)a = \textstyle\int_{b\in\mathbb{G}} (F(a\cdot b))/(Gb) = \int_{b\in\mathbb{G}} \int_{m\in\mathbf{Fin}} [[-, (Gb)m], F(a\cdot b)m],$$

where $[A, B]$ denotes functions between sets.

B BASICS OF FIBRATIONS

Let $P : \mathscr{T} \longrightarrow \mathscr{B}$ be any functor. An arrow $f : X \longrightarrow Y$ in $\mathscr{T}$ is said to be *cartesian* if for every $g : Z \longrightarrow Y$ such that $Pg = Pf \cdot w$ with some $w : PZ \longrightarrow PX$ in $\mathscr{B}$, there is a unique $h : Z \longrightarrow X$ in $\mathscr{T}$ satisfying $Ph = w$ and $f \cdot h = g$:

$$\begin{tikzcd}
PZ \arrow[d, "w"'] \arrow[dr, "Pg"] & & \text{in } \mathscr{B} & \Longrightarrow & Z \arrow[d, dashed, "h"'] \arrow[dr, "g"] & & \text{in } \mathscr{T} \\
PX \arrow[r, "Pf"'] & PY & & & X \arrow[r, "f"'] & Y &
\end{tikzcd}$$

A *split fibration over* $\mathscr{B}$ is a functor $P : \mathscr{T} \longrightarrow \mathscr{B}$ equipped with a mapping κ, called a *split cleavage*, sending every pair $\langle Y, u \rangle$ of an arrow $u : I \longrightarrow J \in \mathscr{B}$ and an object $Y \in \mathscr{T}$ with $PY = J$ to a cartesian morphism $\kappa(Y, u) : X \longrightarrow Y$ in $\mathscr{T}$ such that $P\kappa(Y, u) = u$. The mapping κ is required to preserve identities and composition: $\kappa(Y, \mathrm{id}_J) = \mathrm{id}_Y$ and $\kappa(Y, u \cdot v) = \kappa(Y, u) \cdot \kappa(X, v)$, where X is the domain of $\kappa(Y, u)$. The category $\mathscr{T}$ is called the *total category* and $\mathscr{B}$ is called the *base category.*

A morphism $\langle F, G\rangle : \langle P, \kappa\rangle \to \langle P', \kappa'\rangle$ of split fibrations is a pair of functors $\langle F, G\rangle$ making the left diagram in (28) commute and preserving the cleavage $F\kappa(Y, u) = \kappa(FY, Gu)$:

$$\begin{tikzcd}
\mathscr{T} \arrow[r, "F"] \arrow[d, "P"'] & \mathscr{T}' \arrow[d, "P'"] \\
\mathscr{B} \arrow[r, "G"'] & \mathscr{B}'
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathscr{T} \arrow[r, bend left, "F"{name=F}] \arrow[r, bend right, "F'"'{name=Fp}] \arrow[d, "P"'] & \mathscr{T}' \arrow[d, "P'"] \\
\mathscr{B} \arrow[r, bend left, "G'"{name=Gp}] \arrow[r, bend right, "G"'{name=G}] & \mathscr{B}'
\arrow[Rightarrow, from=F, to=Fp, "\sigma"]
\arrow[Rightarrow, from=G, to=Gp, "\tau"']
\end{tikzcd}
\tag{28}$$

Split fibrations and morphisms form a category $\mathbf{Fib}_s$ with the identities $\langle \mathrm{Id}, \mathrm{Id}\rangle$ and composition $\langle F, G\rangle \circ \langle F', G'\rangle = \langle F \circ F', G \circ G'\rangle$. The category $\mathbf{Fib}_s$ can be extended to a 2-category, in which a 2-cell is a pair $\langle \sigma : F \to F', \tau : G \to G'\rangle$ such that $P' \circ \sigma = \tau \circ P$ as in the right diagram in (28).

Definition B.1 (Grothendieck constructions). Given a functor $F : \mathscr{B}^{op} \to \mathbf{CAT}$, the *Grothendieck construction* yields a split fibration $P : \int F \to \mathscr{B}$ where the category $\int F$ has as objects tuples $\langle I \in \mathscr{B}, a \in FI\rangle$, and arrows $\langle I, a\rangle \to \langle J, a'\rangle$ 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 $\langle \mathrm{id}, \mathrm{id}\rangle$ 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 $\langle u, \mathrm{id}\rangle : \langle J, (Fu)a\rangle \to \langle I, a\rangle$.

A standard result is that the Grothendieck construction is a (2-)equivalence between split fibrations $\mathbf{Fib}_s(\mathscr{B})$ and functors $[\mathscr{B}^{op}, \mathbf{CAT}]$. In fact, it works more generally between pseudofunctors $\mathscr{B}^{op} \to \mathscr{B}$ and (not necessarily split) fibrations.

Definition B.2. With the $\mathbf{CAT}$-functor formulation of modular models, a *lifting* l for a modular model M is a family $l_{\tilde{\Sigma}}$ of natural transformations for each $\tilde{\Sigma} \in \mathcal{F}$:

$$\begin{tikzcd}
\tilde{\Sigma}\text{-}\mathbf{Alg} \arrow[rr, "M_{\tilde{\Sigma}}"] \arrow[dr, "\mathrm{Id}"'] & \arrow[d, phantom, "\overset{l_{\tilde{\Sigma}}}{\Longrightarrow}"] & (\tilde{\Sigma} + \tilde{\Gamma})\text{-}\mathbf{Alg} \arrow[dl, "\pi_1"] \\
& \tilde{\Sigma}\text{-}\mathbf{Alg} &
\end{tikzcd}$$

where π_1 is the projection functor ι_1-$\mathbf{Alg}$, and it is required that for all functorial translations $T : \tilde{\Sigma} \to \tilde{\Phi}$ in $\mathcal{F}$ and all $\langle A, \alpha\rangle \in \tilde{\Phi}$-$\mathbf{Alg}$, it holds that in $\tilde{\Sigma}$-$\mathbf{Alg}$,

$$\pi_1 M_{T,\langle A,\alpha\rangle} \cdot l_{\tilde{\Sigma},\ T\langle A,\alpha\rangle} = T l_{\tilde{\Phi},\ \langle A,\alpha\rangle} : T\langle A, \alpha\rangle \to T\pi_1 M_{\tilde{\Phi}}\langle A, \alpha\rangle$$

C PROOFS

This section contains detailed proofs or sketches for the claims in the paper.

Lemma C.1. *for a cocomplete category $\mathscr{C}$, every functorial translation $T : \hat{\Sigma} \to \hat{\Gamma}$ in $\mathbf{Eqs}_c(\mathscr{C})$ induces a functor $\tilde{T} : \Gamma$-$\mathbf{Alg} \to \Sigma$-$\mathbf{Alg}$ such that it restricts to $T : \hat{\Gamma}$-$\mathbf{Alg} \to \hat{\Sigma}$-$\mathbf{Alg}$ on $\hat{\Gamma}$-$\mathbf{Alg} \subseteq \Gamma$-$\mathbf{Alg}$.*

Proof of Lemma C.1. Given any Γ-algebra $\langle A, \alpha : \Gamma A \to A\rangle$, there is a free $\hat{\Gamma}$-algebra over A: $o : \Gamma(\mathbf{F}_{\hat{\Gamma}}A) \to \mathbf{F}_{\hat{\Gamma}}A$, which T maps to a $\hat{\Sigma}$-algebra $To : \Sigma(\mathbf{F}_{\hat{\Gamma}}A) \to (\mathbf{F}_{\hat{\Gamma}}A)$. We now define $\tilde{T}$ as

$$\tilde{T}\langle A, \alpha\rangle = \langle A,\ \Sigma A \xrightarrow{\Sigma\eta_A} \Sigma(\mathbf{F}_{\hat{\Gamma}}A) \xrightarrow{To} \mathbf{F}_{\hat{\Gamma}}A \xrightarrow{\epsilon_{A,\alpha}} A\rangle.$$

To see that $\tilde{T}$ restricts to T on $\hat{\Gamma}$-$\mathbf{Alg}$, suppose $\langle A, \alpha\rangle \in \hat{\Gamma}$. The functorial translation maps the counit $\epsilon_{A,\alpha} : \langle \mathbf{F}_{\hat{\Gamma}}A, o\rangle \to \langle A, \alpha\rangle$ to a commutative diagram:

$$\begin{tikzcd}
\Sigma(\mathbf{F}_{\hat{\Gamma}}A) \arrow[r, "To"] \arrow[d, "\Sigma\epsilon"'] & \mathbf{F}_{\hat{\Gamma}}A \arrow[d, "\epsilon"] \\
\Sigma A \arrow[r, "T\alpha"'] & A
\end{tikzcd}$$

Therefore $\tilde{T}$ maps α to the algebra $\epsilon \cdot To \cdot \Sigma\eta_A = T\alpha \cdot \Sigma\epsilon \cdot \Sigma\eta_A = T\alpha$ since $\epsilon_{A,\alpha} \cdot \eta_A = \mathrm{id}$. □

Theorem 3.12. *For cocomplete $\mathscr{C}$, every functorial translation $T : \hat{\Sigma} \to \hat{\Gamma}$ in $\mathbf{Eqs}_c(\mathscr{C})$ as a functor $T : \hat{\Gamma}\text{-}\mathbf{Alg} \to \hat{\Sigma}\text{-}\mathbf{Alg}$ has a left adjoint $F : \hat{\Sigma}\text{-}\mathbf{Alg} \to \hat{\Gamma}\text{-}\mathbf{Alg}$.*

Proof of Theorem 3.12. The idea of the proof is essentially the same as the standard result that the free algebra for a functor $\Sigma : \mathscr{C} \to \mathscr{C}$ over an object $A \in \mathscr{C}$ has the same carrier as the initial algebra of the functor $A + \Sigma-$, except that we need to take equations into account.

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

$$\hat{\Gamma}_A \coloneqq \hat{\Gamma} \urcorner_{\mathrm{op}} \mathbf{K}_A \urcorner_{\mathrm{eq}} (\mathbf{K}_{\Sigma A} \vdash L' = R') \tag{29}$$

where $\mathbf{K}_X : \mathscr{C} \to \mathscr{C}$ is the constant functor to X, and $L', R' : (\Gamma + \mathbf{K}_A)\text{-}\mathbf{Alg} \to \mathbf{K}_{GA}\text{-}\mathbf{Alg}$ are

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

where $\tilde{T} : \Gamma\text{-}\mathbf{Alg} \to \Sigma\text{-}\mathbf{Alg}$ is induced by $T : \hat{\Gamma}\text{-}\mathbf{Alg} \to \hat{\Sigma}\text{-}\mathbf{Alg}$ by Lemma C.1. Since constant functors $\mathbf{K}_A$ and $\mathbf{K}_{GA}$ trivially preserves all colimits, the equational system $\hat{\Gamma}_A$ is in $\mathbf{Eqs}_c(\mathscr{C})$ and thus has an initial algebra $\langle B_0, \beta : \Gamma B_0 \to B_0, i : A \to B_0\rangle$.

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

$$\begin{tikzcd} \Sigma A \arrow[r, "\Sigma i"] \arrow[d, "\alpha"'] & \Sigma B \arrow[d, "\tilde{T}\langle B,\beta\rangle"] \\ A \arrow[r, "i"] & B \end{tikzcd} \tag{30}$$

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

It remains to 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 $T : \hat{\Gamma}\text{-}\mathbf{Alg} \to \hat{\Sigma}\text{-}\mathbf{Alg}$. For every $\langle C, \delta\rangle \in \hat{\Gamma}\text{-}\mathbf{Alg}$ and an arrow $f : \langle A, \alpha\rangle \to T\langle C, \delta\rangle$, we need to find a unique $h : \langle B_0, \beta\rangle \to \langle C, \delta\rangle$ such that $Th \cdot i = f$:

$$\begin{tikzcd} \langle A, \alpha\rangle \arrow[r, "i"] \arrow[dr, "f"'] & T\langle B_0, \beta\rangle \arrow[d, "Th"] & & \langle B_0, \beta\rangle \arrow[d, dashed, "h"] \\ & T\langle C, \delta\rangle & & \langle C, \delta\rangle \end{tikzcd}$$

Recall that by definition the translation T preserves carriers and homomorphisms, so $T\langle C, \delta\rangle = \langle C, \delta'\rangle$ for some $\delta' : \Sigma C \to C$. We observe that $\langle C, \delta, f\rangle$ is a model of $\hat{\Gamma}_A$ (29), and thus by the initiality of $\langle B_0, \beta\rangle$, there is a $h : \langle B_0, \beta, i\rangle \to \langle C, \delta, f\rangle$. Since h is a $\hat{\Gamma}_A$-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 $h : \langle B_0, \beta\rangle \to \langle C, \delta\rangle \in \hat{\Gamma}\text{-}\mathbf{Alg}$ with $h \cdot i = f$. Assuming there is such an h', then h' is also a $\hat{\Gamma}_A$-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 $\hat{\Sigma}$-algebra $\langle A, \alpha\rangle$, there is a universal arrow from $\langle A, \alpha\rangle$ to $T : \hat{\Gamma}\text{-}\mathbf{Alg} \to \hat{\Sigma}\text{-}\mathbf{Alg}$. This extends to an adjunction $F \dashv T$ by [Mac Lane 1998, §IV.1 Theorem 2]. □

Theorem 3.13. *The category $\mathbf{Eqs}_c(\mathscr{C})$ is cocomplete if $\mathscr{C}$ is cocomplete.*

Proof of Theorem 3.13. Arbitrary colimits can be constructed from coproducts and coequalisers. Thus it is sufficient to show that $\mathbf{Eqs}_c(\mathscr{C})$ has set-indexed coproducts and coequalisers.

(i) The coproduct of a set of equational systems is obtained by taking the coproduct of signatures and equations. Precisely, the coproduct $\coprod_{i\in I}\hat{\Sigma}_i$ has signature $\coprod_i \Sigma_i$ and equation $\coprod_{i\in I}\Gamma_i \vdash L = R$ where L and $R : (\coprod_{i\in I}\Sigma_i)\text{-}\mathbf{Alg} \to (\coprod_{i\in I}\Gamma_i)\text{-}\mathbf{Alg}$ are

$$L\langle A, \alpha : \textstyle\coprod_{i\in I}\Sigma_i A \to A\rangle = [L_i(\alpha\cdot\iota_i)]_{i\in I} \quad\text{and}\quad R\langle A, \alpha : \textstyle\coprod_{i\in I}\Sigma_i A \to A\rangle = [R_i(\alpha\cdot\iota_i)]_{i\in I}.$$

(ii) Let $T_1, T_2 : \hat{\Gamma} \to \hat{\Sigma}$ be a pair of translations. The codomain $\hat{\Sigma}'$ of their coequaliser is $\hat{\Sigma}$ extended with an additional equation $\tilde{T}_1 = \tilde{T}_2$, where $\tilde{T}_1, \tilde{T}_2 : \Sigma\text{-}\mathbf{Alg} \to \Gamma\text{-}\mathbf{Alg}$ are obtained from T_1 and T_2 by Lemma C.1. The coequaliser $\hat{\Sigma} \to \hat{\Sigma}'$ is the inclusion functor: $\hat{\Sigma}'\text{-}\mathbf{Alg} \hookrightarrow \hat{\Sigma}\text{-}\mathbf{Alg}$. □

Lemma 3.14. *Let $\hat{\Sigma} \in \mathbf{Eqs}(\mathscr{C})$ for $\mathscr{C}$ a category with finite coproducts, and for $i \in \{1,2\}$, let $\Phi_i : \mathscr{C} \to \mathscr{C}$ be a functorial signature and $E_i = (\Psi_i \vdash L_i = R_i)$ be an equation. 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:*

$$\begin{tikzcd}
\hat{\Sigma} \arrow[r, "T_2"] \arrow[d, "T_1"'] & \hat{\Sigma} \mathrel{\lhd_{\mathrm{op}}} \Phi_2 \mathrel{\lhd_{\mathrm{eq}}} E_2 \arrow[d] \\
\hat{\Sigma} \mathrel{\lhd_{\mathrm{op}}} \Phi_1 \mathrel{\lhd_{\mathrm{eq}}} E_1 \arrow[r] & \hat{\Sigma} \mathrel{\lhd_{\mathrm{op}}} (\Phi_1 + \Phi_2) \mathrel{\lhd_{\mathrm{eq}}} E
\end{tikzcd}$$

where $E = (\Psi_1 + \Psi_2 \vdash [L_1 \circ \alpha_1, L_2 \circ \alpha_2] = [R_1 \circ \alpha_1, R_2 \circ \alpha_2])$ and $\alpha_i : (\Sigma + (\Phi_1 + \Phi_2))\text{-}\mathbf{Alg} \to (\Sigma + \Phi_i)\text{-}\mathbf{Alg}$ is the projection functor.

Proof of Lemma 3.14. First of all, there are evident inclusion translations P_i (which are the unlabelled arrows in the pushout diagram):

$$P_i : (\hat{\Sigma} \mathrel{\lhd_{\mathrm{op}}} (\Phi_1 + \Phi_2) \mathrel{\lhd_{\mathrm{eq}}} E')\text{-}\mathbf{Alg} \quad\to\quad (\hat{\Sigma} \mathrel{\lhd_{\mathrm{op}}} \Phi_i \mathrel{\lhd_{\mathrm{eq}}} E_i)\text{-}\mathbf{Alg}$$

such that $T_1 \circ P_1 = T_2 \circ P_2$. Now for every equational system $\hat{\Gamma} \in \mathbf{Eqs}(\mathscr{C})$ with translations functors

$$Q_i : \hat{\Gamma}\text{-}\mathbf{Alg} \to (\hat{\Sigma} \mathrel{\lhd_{\mathrm{op}}} \Phi_i \mathrel{\lhd_{\mathrm{eq}}} E_i)\text{-}\mathbf{Alg}$$

such that $T_1 \circ Q_1 = T_2 \circ Q_2$, we can define a translation functor

$$U : \hat{\Gamma}\text{-}\mathbf{Alg} \to (\hat{\Sigma} \mathrel{\lhd_{\mathrm{op}}} (\Phi_1 + \Phi_2) \mathrel{\lhd_{\mathrm{eq}}} E')\text{-}\mathbf{Alg}$$

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

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

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

Theorem 4.3. *For a cocordial monoidal category $\mathscr{E}$, there is an equivalence $\mathbf{ALG}(\mathscr{E}) \cong \mathbf{Mon}(\mathscr{E})$ between the category $\mathbf{ALG}(\mathscr{E})$ and the category $\mathbf{Mon}(\mathscr{E})$ of monoids in $\mathscr{E}$.*

Proof of Theorem 4.3. The direction $\mathbf{ALG}(\mathscr{E}) \to \mathbf{Mon}(\mathscr{E})$ of the equivalence sends every theory $\tilde{\Sigma} = \langle\hat{\Sigma}, T_\Sigma\rangle$ in $\mathbf{ALG}(\mathscr{E})$ to its initial algebra $\langle\mu\hat{\Sigma}, \alpha^\Sigma\rangle$ regarded as a monoid $T_\Sigma\langle\mu\hat{\Sigma}, \alpha^\Sigma\rangle$. To extend the mapping to a functor, every translation $T : \tilde{\Sigma} \to \tilde{\Gamma} \in \mathbf{ALG}(\mathscr{E})$ induces the unique $\hat{\Sigma}$-homomorphism out of the initial algebra:

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

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

$$T_\Sigma h : T_\Sigma\langle\mu\hat{\Sigma}, \alpha^\Sigma\rangle \to T_\Sigma(T\langle\mu\hat{\Gamma}, \alpha^\Gamma\rangle) = T_\Gamma\langle\mu\hat{\Gamma}, \alpha^\Gamma\rangle$$

The equality in the codomain $T_\Sigma \circ T = T_\Gamma : \hat{\Gamma}\text{-}\mathbf{Alg} \to \mathbf{Mon}(\mathscr{E})$ is by the definition of arrows in $\mathbf{ALG}(\mathscr{E})$.

For the other direction, every monoid $\hat{M} = \langle M, \mu^M, \eta^M \rangle$ in $\mathscr{E}$ is sent to the theory $\hat{M}$-**Act** of *$\hat{M}$-actions on monoids*, which is the theory of $(M \,\Box\, -)$-**Mon** extended with equations

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

saying that $\mathsf{op} : M \,\Box\, \tau \to \tau$ is a monoid action on τ. Every monoid morphism $f : \hat{M} \to \hat{N}$ is mapped to the translation $\hat{M}$-**Act** $\to \hat{N}$-**Act** sending $\hat{N}$-actions (note the contra-variance of translations)

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

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

It remains to show that the mappings above are a pair of equivalence:

- Starting from a monoid $\hat{M}$, it can be shown that the category $(\hat{M}$-**Act**$)$-**Alg** is just the coslice category $\hat{M}/\mathbf{Mon}(\mathscr{E})$ [Fiore and Saville 2017, Proposition 5.5]. Thus the initial algebra of $\hat{M}$-**Act** is $\hat{M}$ as required.
- Starting from a theory $\langle \hat{\Sigma}, T_\Sigma \rangle \in \mathbf{ALG}(\mathscr{E})$ where

$$\hat{\Sigma} = (S \,\Box\, -)\text{-}\mathbf{Mon} \,\Lsh_{\mathrm{eq}}\, (\mathbf{K}_B \vdash L = R),$$

it is mapped to its initial algebra $\mu\hat{\Sigma}$, which is then mapped back to the theory $\mu\hat{\Sigma}$-**Act** with the inclusion translation. We need to construct an isomorphism translation $T : \hat{\Sigma} \to \mu\hat{\Sigma}$-**Act** that preserves monoid operations. Given a monoid A with $\alpha : \mu\hat{\Sigma} \,\Box\, A \to A$ satisfying the laws of $\mu\hat{\Sigma}$-**Act**, T maps it to the $\hat{\Sigma}$-algebra on A with operation

$$S \,\Box\, A \xrightarrow{S \Box \eta^{\mu\hat{\Sigma}}} S \,\Box\, \mu\hat{\Sigma} \,\Box\, A \xrightarrow{\alpha^{\mu\hat{\Sigma}}} \mu\hat{\Sigma} A \xrightarrow{\alpha} A$$

where $\alpha^{\mu\hat{\Sigma}} : S \,\Box\, \mu\hat{\Sigma} \to \mu\hat{\Sigma}$ is the structure map of the initial algebra. For the inverse of T, every tuple $\langle A, \beta : S \,\Box\, A \to A \rangle \in \hat{\Sigma}$-**Alg** is mapped to the following $\mu\hat{\Sigma}$-**Act**-algebra on A:

$$\mu\hat{\Sigma} \,\Box\, A \xrightarrow{(\!|\beta|\!) \Box A} A \,\Box\, A \xrightarrow{\mu^A} A$$

where $(\!|\beta|\!)$ is the unique homomorphism from the initial $\hat{\Sigma}$-algebra $\mu\hat{\Sigma}$ to the $\hat{\Sigma}$-algebra $\langle A, \beta \rangle$.

□

Lemma C.2. *Let $\mathscr{E}$ be a cocordial monoidal category, and $\tilde{\Gamma} \in$ **ALG**$(\mathscr{E})$ and $\tilde{\Sigma} \in$ **Mon/Eqs**$_c(\mathscr{E})$. Then* **Mon/Eqs**$_c(\tilde{\Gamma}, \tilde{\Sigma})$ *is in natural bijection to monoid morphisms $\mu\tilde{\Gamma} \to \mu\tilde{\Sigma}$.*

Proof of Lemma C.2. For one direction of the bijection ϕ, given a translation $T : \tilde{\Gamma} \to \tilde{\Sigma}$, T sends the initial algebra $\mu\tilde{\Sigma}$ of $\tilde{\Sigma}$ to a $\tilde{\Gamma}$-algebra carried by $\mu\tilde{\Sigma}$. Then by the initiality of $\mu\tilde{\Gamma}$, there is a $\tilde{\Gamma}$-homomorphism, which is also a monoid morphism, $u : \mu\tilde{\Gamma} \to \mu\tilde{\Sigma}$. We set the $\phi(T) = u$.

For the back direction of the bijection ϕ, given a monoid morphism $h : \mu\tilde{\Gamma} \to \mu\tilde{\Sigma}$. We define a translation $T : \tilde{\Gamma} \to \tilde{\Sigma}$, i.e. a functor $T : \tilde{\Sigma}$-**Alg** $\to \tilde{\Gamma}$-**Alg**. Recall that $\tilde{\Gamma} \in$ **ALG**$(\mathscr{E})$ must be of the form $\hat{\Gamma} = (G \,\Box\, -)$-**Mon** $\Lsh_{\mathrm{eq}}$ $(\mathbf{K}_B \vdash L = R)$, for some $G \in \mathscr{E}$. The functor T maps every $\tilde{\Sigma}$-algebra $\langle A, \alpha : \Sigma A \to A \rangle$ to the $\hat{\Gamma}$-algebra carried by A with

$$G \,\Box\, A \xrightarrow{G \Box \eta^{\mu\tilde{\Gamma}}} G \,\Box\, \mu\tilde{\Gamma} \,\Box\, A \xrightarrow{\alpha^{\mu\tilde{\Gamma}}} \mu\tilde{\Gamma} \,\Box\, A \xrightarrow{h} \mu\tilde{\Sigma} \,\Box\, A \xrightarrow{(\!|\alpha|\!)} A \,\Box\, A \xrightarrow{\mu^A} A$$

where $\alpha^{\mu\tilde{\Gamma}} : G \,\Box\, \mu\tilde{\Gamma} \to \mu\tilde{\Gamma}$ is the structure map of the initial $\tilde{\Gamma}$-algebra, $(\!|\alpha|\!) : \mu\tilde{\Sigma} \to A$ is the unique $\tilde{\Sigma}$-homomorphism from the initial algebra $\mu\tilde{\Sigma}$ to $\langle A, \alpha \rangle$.

It can be checked that ϕ is a bijection. □

Theorem 4.4. *Let $\mathscr{E}$ be a cocordial monoidal category. (i) The category* $\mathbf{ALG}(\mathscr{E})$ *is a coreflective subcategory of* $\mathbf{Mon/Eqs}_c(\mathscr{E})$*, i.e. there is an adjunction* $\mathbf{ALG}(\mathscr{E}) \overset{\perp}{\underset{\lfloor - \rfloor}{\leftrightarrows}} \mathbf{Mon/Eqs}_c(\mathscr{E})$*. (ii) Moreover, the coreflector $\lfloor - \rfloor$ preserves initial algebras: for every $\langle \hat{\Sigma} \in \mathbf{Eqs}_c(\mathscr{E}), T : \mathbf{Mon} \to \hat{\Sigma}\rangle$, the initial $\hat{\Sigma}$-algebra (viewed as a monoid using T) is isomorphic to the initial algebra of $\lfloor \langle \hat{\Sigma}, T\rangle \rfloor$ as monoids.*

Proof of Theorem 4.4. For part (i) of the theorem, every theory $\langle \hat{\Sigma}, T_\Sigma \rangle \in \mathbf{Mon/Eqs}_c(\mathscr{E})$ always has an initial algebra carried by $\mu\hat{\Sigma} \in \mathscr{E}$ by Theorem 3.13, and $\mu\hat{\Sigma}$ carries a monoid structure by the translation $T_\Sigma : \mathbf{Mon} \to \hat{\Sigma}$. The coreflector $\lfloor - \rfloor$ maps the theory $\langle \hat{\Sigma}, T_\Sigma \rangle$ to $\mu\hat{\Sigma}$-$\mathbf{Act} \in \mathbf{ALG}(\mathscr{E})$ as in the proof of Theorem 4.3. For every theory $\langle \hat{\Gamma}, T_\Gamma \rangle \in \mathbf{ALG}(\mathscr{E})$, by Lemma C.2, each translation in the hom-set $\mathbf{Mon/Eqs}_c(\langle \hat{\Gamma}, T_\Gamma \rangle, \langle \hat{\Sigma}, T_\Sigma \rangle)$ is equivalently a monoid morphism $\mu\hat{\Gamma} \to \mu\hat{\Sigma}$, which is also equivalently a translation in $\mathbf{ALG}(\langle \hat{\Gamma}, T_\Gamma \rangle, \lfloor \langle \hat{\Sigma}, T_\Sigma \rangle \rfloor)$ by Theorem 4.3.

For part (ii) of the theorem, the coreflector maps each $\langle \hat{\Sigma}, T \rangle \in \mathbf{Mon/Eqs}_c(\mathscr{E})$ to the theory $\mu\hat{\Sigma}$-$\mathbf{Act}$. It is not difficult to show that the category algebras of $\mu\hat{\Sigma}$-$\mathbf{Act}$ is equivalent to the coslice category $T\mu\hat{\Sigma}/\mathbf{Mon}(\mathscr{E})$ of monoids under $\mu\hat{\Sigma}$. So the initial algebra of $\mu\hat{\Sigma}$-$\mathbf{Act}$ is still the same monoid $\mu\hat{\Sigma}$. ◻

Definition C.3. For a category $\mathscr{C}$ and two (strict) functors $F, G : \mathscr{C} \to \mathbf{CAT}$, a *lax transformation* $\alpha : F \to G$ consists of a family of functors $\alpha_X : FX \to GX$ for every $X \in \mathscr{C}$ and for every $f : X \to Y$ in $\mathscr{C}$, a natural transformation $\alpha_f : Gf \circ \alpha_X \to \alpha_Y \circ Ff$:

$$\begin{tikzcd}
FX \arrow[r, "Ff"] \arrow[d, "\alpha_X"'] & FY \arrow[d, "\alpha_Y"] \\
GX \arrow[r, "Gf"'] \arrow[ur, Rightarrow, "\alpha_f"] & GY
\end{tikzcd}$$

such that $\alpha_{\mathrm{id}_X} = \mathrm{id} : \alpha_X \to \alpha_X$ for all $X \in C$, and for all $f : X \to Y$ and $g : Y \to Z$,

$$\begin{tikzcd}
FX \arrow[r, "Ff"] \arrow[d, "\alpha_X"'] & FY \arrow[r, "Gg"] \arrow[d, dashed, "\alpha_Y"] & FZ \arrow[d, "a_Z"] \\
GX \arrow[r, "Gf"'] \arrow[ur, Rightarrow, "\alpha_f"] & GY \arrow[r, "Gg"'] \arrow[ur, Rightarrow, "\alpha_g"] & GZ
\end{tikzcd}
=
\begin{tikzcd}
FX \arrow[r, "Ff"] \arrow[d, "\alpha_X"'] & FY \arrow[r, "Gg"] & FZ \arrow[d, "a_Z"] \\
GX \arrow[r, "Gf"'] \arrow[urr, Rightarrow, "\alpha_{g \cdot f}"] & GY \arrow[r, "Gg"'] & GZ
\end{tikzcd}$$

An *oplax* transformation from F to G is similar as above except that the direction of the 2-cells α_f become $\alpha_Y \circ Ff \to Gf \circ Gf$.

Lemma C.4. *For a category $\mathscr{C}$ and two strict functors $F, G : \mathscr{C}^{\mathrm{op}} \to$* $\mathbf{CAT}$*, oplax transformations $F \to G$ are in bijection with functors $\int F \to \int G$ in the over category* $\mathbf{CAT}/\mathscr{C}$*.*

Proof of Lemma C.4. Denote by $p : \int F \to \mathscr{C}$ and $q : \int G \to \mathscr{C}$ the split fibrations obtained from F and G by the Grothendieck construction (Definition B.1).

For one direction, given an oplax transformation $\alpha : F \to G$, we define a functor $K : \int F \to \int G$:

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 arrows K sends every arrow $\langle f : I \to J, g : a \to (Ff)b \rangle : \langle I, a \rangle \to \langle J, b \rangle$ in $\int F$ to $\langle f, g' \rangle$ where g' is the following vertical arrow 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 is well defined 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 an arrow $\langle f, \mathrm{id} \rangle : \langle I, (Ff)b \rangle \to \langle J, b \rangle$ in the total category $\int F$, and this arrow is mapped by K to some arrow $\langle f, g \rangle : \langle I, \alpha_I(Ffb) \rangle \to \langle J, \alpha_J b \rangle$, 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 $\int F$:

$$\begin{tikzcd}
I, Ffb \arrow[r, "f{,}id"] \arrow[d, "id{,}Ffh"'] & J, b \arrow[d, "id{,}h"] \\
I, Ffb' \arrow[r, "f{,}id"'] & J, b'
\end{tikzcd}$$

and K maps this diagram to $\int G$, which is the back square in

$$\begin{tikzcd}
I, \alpha_I(Ff)b \arrow[rr, "K\langle f{,}\mathrm{id}\rangle"] \arrow[dr, "\mathrm{id}{,}\alpha_f b"'] \arrow[dd, "\mathrm{id}{,}\alpha_I(Ff)h"'] & & J, \alpha_J b \arrow[dd, "\mathrm{id}{,}\alpha_J h"] \\
& I, Gf\alpha_J b \arrow[ur, "f{,}\mathrm{id}"'] \arrow[dd, "\mathrm{id}{,}Gf(\alpha_J h)"' near start] & \\
I, \alpha_I(Ff)b' \arrow[rr, "K\langle f{,}\mathrm{id}\rangle"' near end] \arrow[dr, "\mathrm{id}{,}\alpha_f b'"'] & & J, \alpha_J b' \\
& I, Gf\alpha_J b' \arrow[ur, "f{,}\mathrm{id}"'] &
\end{tikzcd}$$

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 commute, which implies the naturality of α_f:

$$\begin{tikzcd}
I, \alpha_I(Ff)b \arrow[dr, "\mathrm{id}{,}\alpha_f b"'] \arrow[dd, "\mathrm{id}{,}\alpha_I(Ff)h"'] & & \\
& I, Gf\alpha_J b \arrow[dd, "\mathrm{id}{,}Gf\alpha_J h"] & \\
I, \alpha_I(Ff)b' \arrow[dr, "\mathrm{id}{,}\alpha_f b'"'] & & J, \alpha_J b' \\
& I, Gf\alpha_J b' \arrow[ur, "f{,}\mathrm{id}"'] &
\end{tikzcd}$$

It can be checked that this natural transformations satisfies the axioms of oplax transformations and that the two directions define a bijection. ◻

Theorem 5.4. *Modular models M of some $\tilde{\Gamma} \in \mathcal{F}$ as in Definition 5.2 are in bijection with functors $\bar{M} : \mathcal{F}$-**Alg** $\to (\mathcal{F} + \tilde{\Gamma})$-**Alg** such that $Q \circ \bar{M} = P$ with P and Q as in (23). A modular model M is strict iff $\bar{M}$ is a split fibration morphism from P to Q, that is, $\bar{M}$ maps arrows $\langle T, \mathrm{id} \rangle$ in P to $\langle T, \mathrm{id} \rangle$ in Q.*

Proof of Theorem 5.4. The bijection between modular models and functors follows from Lemma C.4. The bijection between strict modular models, which are natural transformations between **CAT**-valued functors, and split fibration morphism is standard [Jacobs 1999, §1.10]. ◻

Theorem 6.1. *For each $\tilde{\Gamma} = \langle \hat{\Gamma}, T_\Gamma \rangle \in$ **ALG**$(\mathscr{E})$ over a cocordial monoidal category $\mathscr{E}$, a functor $H : \mathbf{Mon}(\mathscr{E}) \to \hat{\Gamma}\text{-}\mathbf{Alg}$ and a $\tau : \mathrm{Id} \to T_\Gamma \circ H$ can be extended to a strict modular model $\bar{M}$ of $\hat{\Gamma}$ such that the diagram on the right below commutes, and $\bar{M}$ has a lifting $\bar{l}_{\langle\langle \hat{\Sigma}, T_\Sigma \rangle,\, A,\, \alpha\rangle} = \tau_{\langle A,\, T_\Sigma\alpha\rangle}$:*

$$\begin{tikzcd}
 & \hat{\Gamma}\text{-}\mathbf{Alg} \arrow[dr, "T_\Gamma"] \arrow[d, phantom, "\Uparrow\tau"] & \\
\mathbf{Mon}(\mathscr{E}) \arrow[ur, "H"] \arrow[rr, "\mathrm{Id}"'] & & \mathbf{Mon}(\mathscr{E})
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathscr{F}\text{-}\mathbf{Alg} \arrow[r, "\bar{M}"] \arrow[d] & (\mathscr{F} + \tilde{\Gamma})\text{-}\mathbf{Alg} \arrow[d] \\
\mathbf{Mon}(\mathscr{E}) \arrow[r, "H"'] & \hat{\Gamma}\text{-}\mathbf{Alg}
\end{tikzcd} \tag{27}$$

where the unlabelled vertical arrows are the evident projection functors.

Proof sketch of Theorem 6.1. For each object $\langle \hat{\Sigma}, T_\Sigma, A, \alpha \rangle$ of $\mathscr{F}$-**Mon** with

$$\hat{\Sigma} = (S \circ -)\text{-}\mathbf{Mon} \urcorner_{\mathrm{eq}} (\mathbf{K}_G \vdash L = R)$$

we define M to send it to a $(\mathscr{F} + \hat{\Gamma})$-algebra with the same carrier of $H\langle A, T_\Sigma\alpha\rangle \in \hat{\Gamma}$-**Alg**. Since $H\langle A, T_\Sigma\alpha\rangle$ already carries a $\hat{\Gamma}$-algebra, we only need to equip it with a $(S \circ -)$-operation. In other words, we need to 'lift' the algebraic operation on A to $H\langle A, T_\Sigma\alpha\rangle$. This can be done with Jaskelioff and Moggi [2010, Theorem 3.4]'s result that *algebraic* operations can be lifted along monoid morphisms, since each component $\tau_{\langle A, T_\Sigma\alpha\rangle}$ is a monoid morphism. Namely, the lifting is

$$\alpha^\sharp = [\![s : S, h : H_A \vdash \mu^H(\tau_A(\alpha_S(s, \eta^A))),\ h) : H_A]\!] \tag{31}$$

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 (31) satisfies the equation $\mathbf{K}_G \vdash L = R$. This follows from the functoriality of $L, R : ((S \circ -) + \Sigma_{\mathbf{Mon}})\text{-}\mathbf{Alg} \to G\text{-}\mathbf{Alg}$, which implies that the following diagrams commute

$$\begin{tikzcd}
G \arrow[r, "L\langle A,\, \alpha\rangle"] \arrow[dr, "L\langle H_A,\, \alpha^\sharp\rangle"'] & A \arrow[d, "\tau_A"] \\
 & H_A
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
G \arrow[r, "R\langle A,\, \alpha\rangle"] \arrow[dr, "R\langle H_A,\, \alpha^\sharp\rangle"'] & A \arrow[d, "\tau_A"] \\
 & H_A
\end{tikzcd}$$

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 checked that the above M (with an evident arrow mapping) and l defined as $l_{\langle\langle \hat{\Sigma}, T_\Sigma\rangle, A, \alpha\rangle} = \tau_{\langle A, T_\Sigma\alpha\rangle}$ satisfy the conditions for being a modular model. ◻

Theorem 6.4. *For each $\langle \hat{\Gamma}, T_\Gamma \rangle \in$ **SCP**$(\mathscr{E})$ over a cocordial and* closed *monoidal category $\mathscr{E}$, a functor $H : \mathbf{Mon}(\mathscr{E}) \to \hat{\Gamma}\text{-}\mathbf{Alg}$ and a natural transformation $\tau : \mathrm{Id} \to T_\Gamma \circ H$ such that there is some $F : \mathscr{E} \to \mathscr{E}$ and $\sigma : \mathrm{Id} \to F$ satisfying $\mathbf{U} \circ T_\Gamma \circ H = F \circ \mathbf{U}$ and $\tau \circ \mathbf{U} = \sigma \circ U$ can be extended to a strict modular model M of $\langle \hat{\Gamma}, T_\Gamma \rangle$ with a lifting l such that (27) commutes and $l_{\langle\langle \hat{\Sigma}, T_\Sigma\rangle,\, A,\, \alpha\rangle} = \tau_{\langle A,\, T_\Sigma\alpha\rangle}$.*

$$\begin{tikzcd}
 & \hat{\Gamma}\text{-}\mathbf{Alg} \arrow[dr, "T_\Gamma"] \arrow[d, phantom, "\Uparrow\tau"] & \\
\mathbf{Mon}(\mathscr{E}) \arrow[ur, "H"] \arrow[rr, "\mathrm{Id}"] \arrow[d, "\mathbf{U}"'] & & \mathbf{Mon}(\mathscr{E}) \arrow[d, "\mathbf{U}"] \\
\mathscr{E} \arrow[rr, bend left, "F"] \arrow[rr, "\mathrm{Id}"'] & \Uparrow\sigma & \mathscr{E}
\end{tikzcd}$$

Proof sketch of Theorem 6.4. Compared to Theorem 6.1, what is essentially new is how scoped operations $\alpha : S \circ A \circ A \to A$ on existing monoids $\langle A, \eta^A, \mu^A \rangle$ are lifted to

$$\mathbf{U}(T_{\Gamma}(H\langle A, \eta^A, \mu^A \rangle)) = FA.$$

The lifting given by Jaskelioff and Moggi [2010] can be denoted as follows, which makes use of the Cayley embedding of monoids $A \to A/A$ (Example 2.2):

$$\frac{\begin{array}{c}\epsilon = (f : A/A \vdash f\ \eta^A : A) \\ \tilde{\alpha} = (s : A \vdash \lambda x.\ \alpha(s, x, \eta^A) : A/A) \qquad g = (a : A \vdash \lambda x.\ \mu^A(a, x) : A/A)\end{array}}{s : S, a : FA, b : FA \vdash \mu((F\epsilon)(\mu^{F(A/A)}(\sigma(\tilde{\alpha}), Fg)), b) : FA}$$

It can be showed that this lifting preserves functorial equations $E \vdash L = R$ satisfied by α with constant contexts E by the same argument as for Theorem 6.1. □

Theorem 6.5. *Let $\mathscr{E}$ be a cocordial category. For any $\tilde{\Gamma}$ in the family $\mathbf{Mon/Eqs}_c(\mathscr{E})$, the family of functors $\mathbf{F}_{\tilde{\Sigma}} : \tilde{\Sigma}$-$\mathbf{Alg} \to (\tilde{\Sigma} + \tilde{\Gamma})$-$\mathbf{Alg}$ can be extended to a (non-strict) modular model.*

Theorem 6.11 (Dependent Combination). *Let $\mathscr{E}$ be a $\langle \mathbf{Endo}_{\kappa}(\mathbf{Set}), \star, \mathrm{Id} \rangle$ and $\mathcal{F}$ be $\mathbf{ALG}(\mathscr{E})$ or $\mathbf{SCP}(\mathscr{E})$. For each $\tilde{\Gamma} \in \mathcal{F}$, every ordinary model $\hat{A} \in \tilde{\Gamma}$-$\mathbf{Alg}$ induces a strict modular model M of $\tilde{\Gamma}$ such that $M_{\tilde{\Sigma}}\langle B, \beta \rangle$ is carried by $A \circ B$, and M has a lifting $l_{\tilde{\Sigma}, \langle B, \beta \rangle} = \eta^A \circ B$.*

Proof sketch of Theorem 6.11. Given two applicative functors A and B, their composition $A \circ B$ can be equipped with an applicative structure $\eta^{A \circ B} = \eta^A \circ \eta^B$ and the following multiplication:

$$\begin{aligned}((A \circ B) \star (A \circ B))n &\cong \int^{m,k} A(Bm) \times A(Bk) \times n^{m \times k} \\ &\xrightarrow{f} \int^{m,k} A(Bm) \times A(Bk) \times (Bn)^{Bm \times Bk} \\ &\to \int^{m',k'} Am' \times Ak \times (Bn)^{m' \times k'} \\ &\cong (A \star A)(Bn) \xrightarrow{\mu^A} A(Bn)\end{aligned}$$

where the step f makes use of the arrow $n^{m \times k} \to (Bn)^{Bm \times Bk}$ that is the transpose of the following:

$$n^{m \times k} \times Bm \times Bk \to \int^{m,k} Bm \times Bk \times n^{m,k} \cong (B \star B)n \xrightarrow{\mu^B} .Bn$$

To lift a scoped operation $\alpha : S \star A \star A \to A$ to $A \circ B$, we need the fact that there is a canonical morphism $s : S \star (A \circ B) \to (S \star A) \circ B$:

$$\begin{aligned}(S \star (A \circ B))n &\cong \int^{m,k} Sm \times A(Bk) \times n^{m \times k} \\ &\to \int^{m,k} Sm \times A(Bk) \times (Bn)^{m \times Bk} \to \int^{m,k'} Sm \times Ak' \times (Bn)^{m \times k'} \cong (S \star A)(Bn)\end{aligned}$$

We define the lifting of α to $A \circ B$ by

$$S \star (A \circ B) \star (A \circ B) \xrightarrow{s} ((S \star A) \circ B) \star (A \circ B) \xrightarrow{(\overline{\alpha} \circ B) \star (A \circ B)} (A \circ B) \star (A \circ B) \xrightarrow{\mu} A \circ B$$

where $\overline{\alpha} = (S \star A \xrightarrow{S \star A \star \eta^A} S \star A \star A \xrightarrow{\alpha} A)$.

To lift a scoped operation $\beta : G \star B \star B \to B$ to $A \star B$, we need the following canonical morphism $t : G \star (A \circ B) \to A \circ (G \star B)$:

$$\begin{aligned}(G \star (A \circ B))n &\cong \int^{m,k} Gm \times A(Bk) \times n^{m \times k} \\ &\to \int^{m,k} A(Gm \times Bk \times n^{m \times k}) \\ &\to \int^{m,k} A((G \star B)n) \\ &\cong A((G \star B)n)\end{aligned}$$

With t we can define the lifting:

$$G \star (A \circ B) \star (A \circ B) \xrightarrow{t} (A \circ (G \star B)) \star (A \circ B) \xrightarrow{(A \circ \overline{\beta}) \star (A \circ B)} (A \circ B) \star (A \circ B) \xrightarrow{\mu} A \circ B$$

where $\overline{\beta} = (G \star B \xrightarrow{S \star G \star \eta^G} G \star B \star B \xrightarrow{\beta} G)$. □