Modular Models of Monoids with Operations

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

Following the principle of *notions of computations as monoids*, we study theories and models of operations on monoids in monoidal categories. On theories, 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 (not necessarily algebraic) operations. We show examples and general constructions of modular models (i) from monoid transformers, (ii) by composition, and (iii) in symmetric monoidal categories.

1 INTRODUCTION

Algebraic theories, also known as equational theories, of monoids with additional operations, called Σ-*monoids* by Fiore et al. [1999], are widely used in the study of programming languages. The initial algebra of such theories models *abstract syntax* of terms of the operations, and the abstract syntax can be interpreted by other *semantic algebras* (*models*) of Σ-monoids following the initiality of the syntax. Applications in various monoidal categories include:

(i) In the category of sets with cartesian products, free monoids, i.e. *lists*, are the fundamental data structure in functional programming, and their universal properties underpin the study of list-processing functions [Bird and de Moor 1997]. Free monoids with additional operations are *trees*, whose monoid unit is the empty tree, and multiplication is substitution of leaf nodes.

(ii) In the category of endofunctors with composition, monoids, i.e. *monads*, are used to model computational effects [Moggi 1989b; Spivey 1990; Wadler 1995]. Particularly, monads arising from equational theories allow computational effects to be specified by their associated effectful operations and equational laws [Plotkin and Power 2002, 2004], and algebras of these monads/theories are designed as a programming construct known as *handlers* [Plotkin and Pretnar 2013].

(iii) In certain monoidal categories, generalisations of monads such as *arrows* [Hughes 2000] and *applicative functors* [Mcbride and Paterson 2008] are also monoids [Pieters et al. 2020; Rivas and Jaskelioff 2017], hence Rivas and Jaskelioff's principle '*notions of computations as monoids*'.

(iv) In the presheaf category $\mathbf{Set}^{\mathbf{Fin}}$ with a certain monoidal structure, free monoids with additional operations are used to model abstract syntax with *variable bindings* [Fiore et al. 1999], for which the monoid multiplication means simultaneous substitution of variables.

A recurring pattern in these examples is that a phenomenon is modelled with an algebraic theory, which induces an initial/syntactic model, and the initial model can be interpreted by other models. This approach is appealing for several reasons: (i) interpretation is compositional by construction; (ii) syntactic structures can be interpreted with different semantic models; (iii) algebraic theories can be combined [Hyland et al. 2006], allowing syntax to be specified in a modular way, known as *data types à la carte* in functional programming [Swierstra 2008].

Lack of Modularity for Models. There is a caveat: although modularity of syntax is attained by combining equational theories, models are *less modular*. Specifically, let $\Sigma_i : \mathscr{E} \to \mathscr{E}$ for $i \in \{1, 2\}$ be two signatures of operations (without equations) over a monoidal category $\mathscr{E}$, and let $\langle A_i \in \mathscr{E}, \mu_i, \eta_i \rangle$ be two monoids equipped with operations $a_i : \Sigma_i A_i \to A_i$ respectively. The two signatures can be combined modularly, for example, by their coproducts $\Sigma_1 + \Sigma_2$, but there seems

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.

no canonical way to combine the two models A_1 and A_2 into a model of the combined signature $\Sigma_1 + \Sigma_2$ in general—it is not clear how to define a nontrivial monoid C_{A_1,A_2} with an operation

$$(\Sigma_1 + \Sigma_2)C_{A_1,A_2} \cong (\Sigma_1 C_{A_1,A_2} + \Sigma_2 C_{A_1,A_2}) \to C_{A_1,A_2}.$$

This problem is practically relevant, since ideally programming language designers would like not only to model the syntax of languages modularly but also to interpret the language modularly, in a way that it is not necessary to modify existing interpreters when adding new language features—traditionally called the *expression problem* [Wadler 1998] or the *stable denotation problem* [Cartwright and Felleisen 1994] in functional programming. This problem is certainly not new and has been treated in several ways, especially in the study of monads and computational effects:

(i) *The compositional approaches* combine two monads A_1 and A_2 (in general, monoids) by their coproducts $A_1 + A_2$ in the category of monads, which can be explicitly characterised in some cases [Adámek et al. 2012; Ghani and Uustalu 2004; Hyland et al. 2006].

(ii) *The incremental approaches*, such as monad transformers [Jaskelioff and Moggi 2010; Moggi 1989a], layered monads [Filinski 1999], turn an existing monad A into a new one TA, equipped with a lifting morphism $A \to TA$ and some new operations on TA.

However, the existing work does not fully resolve the modularity problem because it is still not clear how to lift existing operations $a_i : \Sigma_i A_i \to A_i$ to the combined monad $A_1 + A_2$ (or the transformed monad TA) in general. It can be done canonically in the special case of Σ being the signature of *algebraic operations*. In this case $\Sigma_i = S_i \circ -$ for some endofunctor S_i, and the operation $S_i \circ A_i \to A_i$ is required to commute with monad multiplication. We can define

$$S_i \circ (A_1 + A_2) \xrightarrow{S_i \circ \eta^{A_i}} S_i \circ A_i \circ (A_1 + A_2) \xrightarrow{a_i} A_i \circ (A_1 + A_2) \xrightarrow{\iota_i} (A_1 + A_2) \circ (A_1 + A_2) \xrightarrow{\mu} (A_1 + A_2)$$

and similarly for the incremental approaches by replacing ι_i with the lifting $A \to TA$.

However, there are many operations in practice that are not algebraic: (i) *higher-order* operations, such as lambda abstraction [Fiore et al. 1999] and explicit substitution [Ghani et al. 2006], do not have the shape $S_i \circ A_i \to A_i$, and (ii) *first-order non-algebraic* operations, called *scoped operations* by Wu et al. [2014], have the shape $S_i \circ A_i \to A_i$ but do not commute with monad multiplication. Typical examples are exception catching and parallel composition of processes.

Overview. The goal of this paper is to develop a categorical framework of *modular models* for studying the aforementioned problem systematically. We first define a concept called *monoidal theory families*, which are categories of theories of monoids with additional operations, such that the family of theories is closed under coproducts, and every theory in the family admits free algebras. Examples of this concept include the family of *algebraic operations*, and the family of *scoped operations*, and the family of *variable-binding operations*. Then a *modular model* M of some theory $\tilde{\Gamma}$ in a theory family $\mathcal{F}$ is a family of functors $\tilde{\Sigma}\text{-Alg} \to (\tilde{\Sigma} + \tilde{\Gamma})\text{-Alg}$ natural in $\tilde{\Sigma} \in \mathcal{F}$. In other words, a modular model *sends monoids with operations to monoids with more operations*.

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

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

and for now let us ignore equations on these operations. The operation *throw* is of course for throwing an exception. The operation *catch* is for catching an exception in a scope: the product $M \times M$ in the argument of *catch* is a pair of computations, one for the program p whose exceptions will be caught, the other one for the program h handling the exception if there is one. The operation *catch* additionally takes an *explicit continuation* argument k, corresponding to the $- \circ M$ in the signature. Thus a call $\mathit{catch}\ (\langle p, h \rangle, k)$ should be understood as handling the exception in p using

h, and then continues as k. The explicit continuation argument is exactly the trick to workaround the limitation of algebraic operations, and we will say more about this later in §4.

More generally, an algebraic operation on a monad M takes the form of $A \circ M \to M$ and a scoped operation takes the form of $B \circ M \circ M \to M$ (for *catch*, $(M \times M) \circ M \cong (\mathrm{Id} + \mathrm{Id}) \circ M \circ M$). We then collect *all theories* of monads with some algebraic and scoped operations as a category $\mathrm{SCP}(\mathscr{E})$, which we call the *monoidal theory family of scoped operations* over $\mathscr{E}$, whose arrows are *translations* of theories. The category $\mathrm{SCP}(\mathscr{E})$ has finite coproducts by taking the coproduct of the signatures and equations. Moreover, each theory in $\mathrm{SCP}(\mathscr{E})$ 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 EC^* of the initial algebra can be characterised as the initial solution to

$$\mathrm{EC}^* \cong \mathrm{Id} + 1 + (\mathrm{EC}^* \times \mathrm{EC}^*) \circ \mathrm{EC}^* \in \mathrm{Endo}_f(\mathbf{Set}).$$

The monad EC^* models *syntactic programs* with exception throwing and catching.

A *modular model* M of the theory EC in $\mathrm{SCP}(\mathscr{E})$ is a family of functors as follows:

$$M_{\tilde{\Sigma}} : \tilde{\Sigma}\text{-Alg} \to (\tilde{\Sigma} + \mathrm{EC})\text{-Alg},$$

natural in $\tilde{\Sigma} \in \mathrm{SCP}(\mathscr{E})$. Here $(-)$-Alg is the functor $\mathrm{SCP}(\mathscr{E}) \to \mathbf{CAT}$ sending each theory to the category of its models. For example, one way to define a modular model of EC is to send every $\tilde{\Sigma}$-algebra carried by A to $A \circ (1 + \mathrm{Id})$ equipped with operations from $\tilde{\Sigma}$ and EC (Example 6.4).

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

$$h : (\tilde{\Sigma} + \mathrm{EC})^* \to M_{\tilde{\Sigma}}(\tilde{\Sigma}^*) \qquad \text{in } (\tilde{\Sigma} + \mathrm{EC})\text{-Alg}$$

which interprets EC but leaves $\tilde{\Sigma}$-operations uninterpreted. To summarise the example, we have generalised free algebras of algebraic operations and their handlers [Plotkin and Pretnar 2013] to non-algebraic operations, in a way that *modularity* of models is maintained (which was not considered in Yang and Wu [2021] and Piróg et al. [2018]'s work on models of scoped effects).

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 we discuss colimits in this category. In §4, we define *monoidal theory families* and study the connections between some important 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. Finally in §7, we discuss related and future work. Along these lines, we make the following technical contributions:

- We show in §3 a syntactic framework to present equational theories over monoidal categories, based on Fiore and Hur [2009]'s *equational system* and a *metalanguage for monoidal categories* by Jaskelioff and Moggi [2010]. And in §3.3, we introduce a notion of *functorial translations* between equational systems and give conditions for the cocompleteness of the category of equational systems and functorial translations.
- We show a general *monoid-theory correspondence* (Theorem 4.3)—under mild conditions on the monoidal category $\mathscr{E}$, the theory family $\mathrm{ALG}(\mathscr{E})$ of monoids with algebraic operations is equivalent to the category $\mathrm{Mon}(\mathscr{E})$ of monoids in $\mathscr{E}$, generalising the classical correspondence between finitary monads and first-order finitary algebraic theories.
- We show that the theory family $\mathrm{ALG}(\mathscr{E})$ of algebraic operations is a coreflective subcategory of the family of (almost) all theories, and the coreflection *preserves initial algebras* (Theorem 4.4). This means that algebraic operations are enough for modelling abstract syntax.

- We show in §6.1 how modular models of algebraic operations and scoped operations can be obtained from monoid transformers based on a result by Jaskelioff and Moggi [2010].
- We show in §6.2 that modular models can be *composed*, justifying the name *modular models*. And we show a general *fusion lemma* (Lemma 6.8), saying that interpreting syntax with two modular model one by one is equal to interpreting with their composite.
- We show in §6.3 constructions of modular models in symmetric monoidal categories, in particular, a modular model of the effect of *staging*.

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 concisely (§2.2).

2.1 Monoidal Categories and Monoids

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

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

satisfying some coherence axioms [Mac Lane 1998, §VII.1]. A monoidal category is *(right) closed* if all functors $- \Box A$ 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* $\mu : M \Box M \to M$ and a *unit* $\eta : I \to M$ making the following diagrams commute:

$$\begin{array}{ccccc} (M \Box M) \Box M & & \xrightarrow{\alpha_{M,M,M}} & & M \Box (M \Box M) \\ {\scriptstyle \mu \Box M}\downarrow & & & & \downarrow{\scriptstyle M \Box \mu} \\ M \Box M & \xrightarrow[\mu]{} & M & \xleftarrow[\mu]{} & M \Box M \end{array} \qquad \begin{array}{ccccc} I \Box M & \xleftarrow{\lambda_M} & M & \xrightarrow{\rho_M} & M \Box I \\ {\scriptstyle \eta \Box M}\downarrow & & \| & & \downarrow{\scriptstyle M \Box \eta} \\ M \Box M & \xrightarrow[\mu]{} & M & \xleftarrow[\mu]{} & M \Box M \end{array} \tag{1}$$

Below we review several monoidal categories which are relevant in programming, especially several monoidal structures on endofunctors, for which monoids model different flavours of *notions of computations* [Jacobs et al. 2009; Pieters et al. 2020; Rivas and Jaskelioff 2017], and they will serve as the main application of the theory of modular models of monoids that we develop in this paper.

Monads. The category $\mathrm{Endo}(\mathscr{C})$ 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 formula 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.

Unfortunately, the category $\mathrm{Endo}(\mathscr{C})$ is usually not as well behaved as we would like for doing algebraic theories on it, even when $\mathscr{C}$ itself is a very nice category such as **Set**. In particular,

Endo(**Set**) is not closed with respect to either cartesian products or 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 $F \in \mathrm{Endo}(\mathbf{Set})$ is called finitary if it preserves *filtered colimits* [Adamek and Rosicky 1994]. A useful characterisation of finitariness is that *we lose no information if we restrict F to finite sets*, in the sense that if we first restrict F as $F \circ V : \mathbf{Fin} \to \mathbf{Set}$ on the full subcategory of finite sets, where $V : \mathbf{Fin} \to \mathbf{Set}$ is the inclusion functor, and then take the left Kan extension of $F \circ V$ along V, the result is still isomorphic to F. In other words, we have the following equivalence, where $\mathrm{Endo}_f(\mathbf{Set}) \subseteq \mathrm{Endo}(\mathbf{Set})$ denotes the full subcategory of finitary endofunctors:

$$\mathrm{Endo}_f(\mathbf{Set}) \underset{-\circ V}{\overset{\mathrm{Lan}_V -}{\underset{\longrightarrow}{\overset{\longleftarrow}{\cong}}}} \mathbf{Set}^{\mathbf{Fin}} \tag{2}$$

The category $\mathrm{Endo}_f(\mathbf{Set})$ inherits the monoidal structure $\langle \circ, \mathrm{Id} \rangle$ of Endo(**Set**). And what is new is that $- \circ F : \mathrm{Endo}_f(\mathscr{C}) \to \mathrm{Endo}_f(\mathscr{C})$ always has a right adjoint for all $F \in \mathrm{Endo}_f(\mathbf{Set})$, so $\langle \mathrm{Endo}_f(\mathbf{Set}), \circ, \mathrm{Id} \rangle$ is closed. Also, the category $\mathbf{Set}^{\mathbf{Fin}}$, and thus $\mathrm{Endo}_f(\mathbf{Set})$, is complete and cocomplete since **Set** is so [Mac Lane 1998, §V.3].

Monoids in $\mathrm{Endo}_f(\mathbf{Set})$ are called *finitary monads* on **Set**, and they are known to be equivalent to (finitary) *Lawvere theories*, or simply known as *algebraic theories* [Linton 1966] and *abstract clones* [Mahmoud 2010]. Computational effects modelled by Lawvere theories are usually called *algebraic effects* [Plotkin and Power 2002, 2004].

Apart from modelling computational effects, another related but slightly different application of monoids in $\mathbf{Set}^{\mathbf{Fin}}$ 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 (2), the monoidal structure $\langle \circ, \mathrm{Id} \rangle$ of $\mathrm{Endo}_f(\mathbf{Set})$ is equivalent to $\langle \bullet, V \rangle$ on $\mathbf{Set}^{\mathbf{Fin}}$ where

$$Vn = n \qquad \text{and} \qquad (F \bullet G)n = \int^{m \in \mathbf{Fin}} (Fm) \times (Gn)^m. \tag{3}$$

So the monoid unit $V \to M$ is then embedding *variables* as M-terms, and the monoid multiplication $M \bullet M \to M$ is *simultaneous substitution* of terms for variables. 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 $\mathrm{Endo}_f(\mathbf{Set}) \cong \mathbf{Set}^{\mathbf{Fin}}$ by any *locally finitely presentable* (lfp) category $\mathscr{C}$ and **Fin** with the subcategory $\mathscr{C}_f$ of *finitely presentable objects* in $\mathscr{C}$. This results in a closed monoidal category $\langle \mathrm{Endo}_f(\mathscr{C}), \circ, \mathrm{Id} \rangle$ that is complete and cocomplete [Adamek and Rosicky 1994]. Other examples of lfp categories include the category of posets, the category of categories, presheaf categories $\mathbf{Set}^{\mathscr{C}}$, and functor categories $\mathscr{C}^{\mathscr{D}}$ for any lfp categories $\mathscr{C}$ and small category $\mathscr{D}$ (so $\mathrm{Endo}_f(\mathbf{Set})$ itself is also lfp since it is equivalent to $\mathbf{Set}^{\mathbf{Fin}}$ and **Set** is lfp). Thus lfp categories provide a very general setting to study algebraic theories, which we will use in this paper.

However, a notable *non*example of lfp categories is the category $\omega\mathbf{Cpo}$ of ω-complete partial orders, which is only locally *countably* presentable but not lfp. This suggests that we should work in general with locally κ-presentable (lκp) categories and functors preserving κ-filtered colimits, for an arbitrary regular cardinal κ. However, we will stick with lfp categories $\mathscr{C}$ in this paper, as it simplifies presentation, but we expect our development generalises for lκp categories straightforwardly.

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 $\mathrm{Endo}_f(\mathscr{C})$ is also cartesian closed whenever $\mathscr{C}$ is lfp and cartesian closed. The cartesian unit and product in $\mathrm{Endo}_f(\mathscr{C})$ are defined pointwise:

$$1n = 1_{\mathscr{C}} \qquad (F \times G)n = Fn \times Gn \qquad (F^G)n = \int_{m \in \mathscr{C}_f} \Pi_{\mathscr{C}(-,m)} (Fm)^{Gm}.$$

The exponential is more involved, given by an end formula above. A computational interpretation of cartesian monoids in $\mathrm{Endo}_f(\mathscr{C})$ is that they model *notions of independent computations*: the cartesian multiplication $M \times M \to M$ in $\mathrm{Endo}_f(\mathscr{C})$ composes two computations that have no dependency and return the same type of values, in contrast to monad multiplication $M \circ M \to M$ that composes two computations in which the second depends on the result of the first one.

Applicatives. Between the two extremes of $M \circ M$ and $M \times M$, there are monoidal structures on $\mathrm{Endo}_f(\mathscr{C})$ 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 $\mathrm{Endo}_f(\mathbf{Set})$ has as unit the identity functor, and the monoidal product is given by the following coend formula:

$$(F \star G)n = \int^{m,k \in \mathbf{Fin} \times \mathbf{Fin}} Fm \times Gk \times n^{m \times k}. \tag{4}$$

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 $- \star F$ given by $F \multimap G = \int_{n \in \mathbf{Fin}} (G(- \times n))^{Fn}$.

Monoids for $\langle \star, \mathrm{Id} \rangle$ are called *applicative functors* or simply *applicatives* [Mcbride and Paterson 2008; Paterson 2012]. Informally, they model *notions of almost independent computations whose results are combined purely*. 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.

Applicatives are a weaker notion than (strong) monads, since intuitively independent computations are special cases of dependent computations. Precisely, for any two functors $F, G \in \mathrm{Endo}_f(\mathbf{Set})$, there are canonical strengths, i.e. natural transformations:

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

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

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

where the first arrow is repeated use of the strengths of F and G, and the second arrow is functions evaluation. Consequently, for any monad $\langle M,\ \mu : M \circ M \to M,\ \eta : \mathrm{Id} \to M \rangle$, it induces an applicative functor with unit η and multiplication $M \star M \xrightarrow{e} M \circ M \xrightarrow{\mu} M$. However, there are many applicative functors that are not obtained from monads in this way [Paterson 2012].

More generally, we can replace **Set** with any *lkp as a cartesian closed* category $\mathscr{V}$ [Kelly 1982] and also the coend in (4) with a $\mathscr{V}$-enriched coend, which allows us to give a more accurate formulation of applicatives in functional languages with general recursion by setting $\mathscr{V} = \omega\mathbf{Cpo}$. However, we will only consider the unenriched setting in this paper for simplicity.

Arrows. Between applicatives and monads, there is another flavour of notions of computations that allows *data dependency* but *not control dependency*, known as *arrows* [Hughes 2000; Lindley et al. 2011]. Unlike monads and applicatives, arrows are not monoids in $\mathrm{Endo}(\mathscr{C})$, but in the category of *strong endoprofunctors*. An endoprofunctor P on a category $\mathscr{C}$ with finite products is just a functor $P : \mathscr{C}^{op} \times \mathscr{C} \to \mathbf{Set}$. Informally, the set $P(a, b)$ is P-computations from type a to type b. A strong endoprofunctor is additionally equipped with a family s of morphisms $s_{abc} : P(a, b) \to P(a \times c, b \times c)$, called a *strength*, that is natural in a, b and dinatural in c satisfying certain coherence conditions

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

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

Fig. 1. Typing rules for the metalanguage

(see e.g. Rivas and Jaskelioff [2017, Def 7.1]). The strength s_{abc} informally means every computation in $P(a, b)$ can also be run alongside some unused data c. Strong endoprofunctors and strength-compatible natural transformations between them form a category $\mathrm{EndoPro}_s(\mathscr{C})$.

The category $\mathrm{EndoPro}_s(\mathscr{C})$ can be equipped with a monoidal structure $\langle I, \otimes \rangle$:

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

where $\mathscr{C}(a, b)$ is the hom-set of $\mathscr{C}$. Both I and $P \otimes Q$ have a canonical strength (see e.g. [Rivas and Jaskelioff 2017, Def 7.2]). Informally, the product $P \otimes Q$ are two computations P and Q with some type of data x flowing from P to Q, so it allows more dependency than applicatives, but unlike $M \circ M$, it does not allow the second computation to *dynamically* depend on the return value of P (see Pieters et al. [2020] and Lindley et al. [2011] for more detailed comparisons).

2.2 A Metalanguage for Monoidal Categories

When the monoidal category gets complex, commutative diagrams like those in (1) can be unwieldy, we will use a typed calculus introduced by Jaskelioff and Moggi [2010] as a *metalanguage* to denote constructions in monoidal categories, just like 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.

Syntax. The syntax of the calculus is as follows:

$$\begin{array}{rl} \text{Types} & A, B ::= \alpha \mid A \boxtimes B \mid \mathsf{I} \\ \text{Terms} & t ::= x \mid f(t) \mid * \mid (t_1, t_2) \mid \mathsf{let}\ * = t_1\ \mathsf{in}\ t_2 \mid \mathsf{let}\ (x_1, x_2) = t_1\ \mathsf{in}\ t_2 \\ \text{Contexts} & \Gamma ::= \cdot \mid \Gamma, x : A \end{array}$$

where α ranges over a set of base types; x ranges over an infinite set of variables; and f ranges over a set of primitive operations, each associated with two types $f : A \to B$.

Typing. The type system in Figure 1 of the calculus is a substructural one, resembling Polakow and Pfenning [1999]'s *intuitionistic non-commutative linear logic*, since the language is to be interpreted 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 $[\![\alpha]\!] \in \mathrm{Ob}(\mathscr{E})$ is assigned to each base type, all types and typing contexts can be interpreted as objects in $\mathscr{E}$:

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

Given an interpretation $[\![f]\!] : [\![A]\!] \to [\![B]\!]$ in $\mathscr{E}$ for each primitive operation $f : A \to B$, all well typed terms $\Gamma \vdash t : A$ can be interpreted as morphisms $[\![t]\!] : [\![\Gamma]\!] \to [\![A]\!]$ inductively. The interpretation

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

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.

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 $\llbracket - \rrbracket$, we write them in the subscript like $\llbracket - \rrbracket_{\{\alpha \mapsto A\}}$.

Example 2.1. Assume a base type M and a primitive operation $\mu : M \square M \longrightarrow M$. The first diagram in the laws of monoids (1) 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, but the equals sign here has no formal meaning):

$$\frac{\mu : M \square M \longrightarrow M}{x : M, y : M, z : M \vdash \mu(\mu(x, y), z) \ = \ \mu(x, \mu(y, z)) : M} \tag{5}$$

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 freely 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 constructor B/A, new syntax $\lambda x : A.\, t$ and $t_1 \ t_2$ with 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:

$$\llbracket \lambda x : A.\, t : B/A \rrbracket = abst(\llbracket t \rrbracket) \qquad \llbracket t_1 \ t_2 \rrbracket = ev \cdot (\llbracket t_1 \rrbracket \square \llbracket t_2 \rrbracket)$$

where $abst : \mathscr{E}(C \square A, B) \longrightarrow \mathscr{E}(C, B/A)$ is the natural isomorphism associated to the adjunction $(- \square A) \dashv (-/A)$ and $ev : (B/A) \square A \longrightarrow B$ 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 (right adjoint to cartesian products).

Other strucutres 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 constructor $\times$ with typing rules:

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

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

$$f : M/M, \ g : M/M \vdash \lambda x.\, f \ (g \ x) : M/M. \tag{6}$$

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 $\llbracket r \rrbracket \cdot \llbracket e \rrbracket = \mathrm{id}_{\llbracket M \rrbracket}$. 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 (6) 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 $\mathscr{E} = \langle \mathrm{Endo}_f(\mathscr{C}), \circ, \mathrm{Id} \rangle$, this optimisation is known as *codensity transformation* [Hinze 2012]. We will also use this fact later for constructing modular models Theorem 6.5.

3 EQUATIONAL SYSTEMS AND TRANSLATIONS

We have seen monoids in various monoidal categories, but if the only thing that we can do with a monoid is its unit and multiplication, then it is barely useful in practice. 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, so that we can systematically 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 (§3.3).

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 $h \cdot \alpha = \beta \cdot \Sigma h$. The forgetful functor dropping the structure map is denoted by $\mathrm{U}_\Sigma : \Sigma\text{-Alg} \to \mathscr{C}$.

Equations of theories are usually presented as commutative diagrams (like the first diagram in (1) for associativity) containing formal arrows $\Sigma A \to A$ for the signature functor Σ. One way to formulate such a diagram is a pair of functors $L, R : \Sigma\text{-Alg} \to \Gamma\text{-Alg}$ 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* $L, R : \Sigma\text{-Alg} \to \Gamma\text{-Alg}$ such that $\mathrm{U}_\Gamma \circ L = \mathrm{U}_\Sigma$ and $\mathrm{U}_\Gamma \circ R = \mathrm{U}_\Sigma$.

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.

Among the algebras of an equational theory $\hat{\Sigma}$[1]over $\mathscr{C}$, the *free algebras* are particularly useful since they represent *abstract syntax* of terms built from operations of the theory. And the abstract syntax can be *interpreted* with another model using the free-forgetful adjunction:

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

Given any model $\langle A, \alpha \rangle$ of $\hat{\Sigma}$ and $g : X \to A$, the morphism $\phi(g) : \mathrm{Free}\, X \to \langle A, \alpha \rangle$ 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 3.2 (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*

$$\hat{\Sigma}\text{-Alg} \underset{\longleftarrow}{\overset{\longleftarrow}{\bot}} \Sigma\text{-Alg} \underset{\longleftarrow}{\overset{\longleftarrow}{\bot}} \mathscr{C}$$

to the inclusion functor $\hat{\Sigma}$-Alg $\to$ Σ-Alg and forgetful functor Σ-Alg $\to \mathscr{C}$ respectively.

Remark. 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.

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):

(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[2] 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 exactly 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 $T : \Sigma\text{-Alg} \to \Gamma\text{-Alg}$ such that

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

By structural induction on typing derivations in the style of Reynolds [1983]'s abstraction theorem, it is possible to show that $U_\Gamma \circ T = U_\Sigma$ as required in Definition 3.1. For example, the two terms (5) denote a pair of functorial terms $(- \,\square\, -)$-Alg $\to$ $(- \,\square\, - \,\square\, -)$-Alg.

[1] 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.
[2] The rule for positivity is standard: 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 constructors.

3.2 Examples of Equational Systems

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

$$\mathrm{Mon} = (\Sigma_{\mathrm{Mon}} \triangleright \Gamma_{\mathrm{Mon}} \vdash L_{\mathrm{Mon}} = R_{\mathrm{Mon}}) \tag{7}$$

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

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

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

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

where the components are as follows, c.f. (1):

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

Notation 3.3. As demonstrated above, although Definition 3.1 only allows 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 $\hat{\Sigma}\ {}^{\curvearrowleft}_{\mathrm{op}}\ \Sigma$ the extension of $\hat{\Sigma}$ with more operations of signature $\Sigma' : \mathscr{C} \to \mathscr{C}$:

$$\hat{\Sigma}\ {}^{\curvearrowleft}_{\mathrm{op}}\ \Sigma \coloneqq (\Sigma + \Sigma' \triangleright \Gamma \vdash L \circ \pi = R \circ \pi)$$

where $\pi : (\Sigma + \Sigma')\text{-Alg} \to \Sigma\text{-Alg}$ is the projection. Similarly, we denote extending $\hat{\Sigma}$ with an equation by $\hat{\Sigma}\ {}^{\curvearrowleft}_{\mathrm{eq}}\ (\Gamma' \vdash L' = R') \coloneqq (\Sigma \triangleright (\Gamma + \Gamma') \vdash [L, L'] = [R, R'])$.

A model of the equational theory Mon is exactly a monoid in $\langle \mathscr{E}, \square, I \rangle$. When $\mathscr{E}$ is cocomplete and $\square : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ preserves ω-chains, then Theorem 3.2 is applicable for the existence of free monoids. Moreover, when $\mathscr{E}$ is closed, there is a simple formula for free monoids: for all $A \in \mathscr{E}$, the free monoid over A is the initial algebra $\mu X.I + A \square X$ equipped with appropriate monoid operations [Fiore 2008]. This formula is very handy in practice: when $\mathscr{E}$ is $\langle \mathbf{Set}, \times, 1 \rangle$, this formula is exactly the usual definition $\mu X.1 + A \times X$ of lists of A-elements; and when $\mathscr{E}$ is $\langle \mathrm{Endo}_f(\mathscr{C}), \circ, \mathrm{Id} \rangle$ or $\langle \mathrm{Endo}_f(\mathscr{C}), \star, \mathrm{Id} \rangle$ in §2.1, the assumptions will also be met, and this gives formulas for free monads and free applicatives that are implementable in programming languages [Rivas and Jaskelioff 2017].

Σ-Monoids. Monoids 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

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

for all X in $\mathscr{E}$ and $\langle Y \in \mathscr{E}, f : I \to Y \rangle$ in the coslice category $I/\mathscr{E}$, satisfying coherence conditions analogous to those of strengths [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 rules

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

Then the equational system Σ-Mon of Σ-*monoids* extends the theory Mon of monoids (7) with a new operation $\mathrm{op} : \Sigma\tau \to \tau$ and a new equation $L_{\Sigma\text{-Mon}} = R_{\Sigma\text{-Mon}}$:

$$\Sigma\text{-Mon} = (\mathrm{Mon} \upharpoonleft_{\mathrm{op}} \Sigma) \upharpoonleft_{\mathrm{eq}} \left((\Sigma -) \square - \vdash L_{\Sigma\text{-Mon}} = R_{\Sigma\text{-Mon}}\right) \tag{8}$$

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

$$x : \Sigma\tau, y : \tau \vdash \mu(\mathrm{op}\ x, y) = \mathrm{op}((\Sigma\mu)(\theta_{\tau,\langle\tau,\eta\rangle}(x, y))) : \tau \tag{9}$$

The special case of (9) for $\Sigma = A \square -$ with pointed strength $(A \square X) \square Y \cong A \square (X \square Y)$ is exactly *algebraicity* of operations $\mathrm{op} : A \square \tau \to \tau$ 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 (9) *generalised algebraicity*.

A model of the equational system Σ-Mon is exactly a Σ-monoid. When the monoidal category $\mathscr{E}$ is cocomplete and functors Σ, Γ, □ all preserve colimits of ω-chains, Theorem 3.2 ensures the existence of free Σ-monoids. When $\mathscr{E}$ is additionally closed, such as $\langle \mathrm{Endo}_f(\mathscr{C}), \circ, \mathrm{Id}\rangle$ and $\langle \mathrm{Endo}_f(\mathbf{Set}), \star, \mathrm{Id}\rangle$ in §2.1, there is again a simple description of the free Σ-monoid: it is carried by the initial algebra $\mu X.I + A \square X + \Sigma X$. This formula has a wide range of applications: modelling the abstract syntax of variable binding [Fiore and Szamozvancev 2022], the syntax of explicit substitution [Ghani et al. 2006], and the syntax of 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 3.4 (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 $\mathscr{E} = \langle \mathrm{Endo}_f(\mathscr{C}), \circ, \mathrm{Id}\rangle$, the equational system ET_E describes (finitary) monads $M : \mathscr{C} \to \mathscr{C}$ equipped with a natural transformation

$$(\textstyle\coprod_E 1) \circ M = \coprod_E (1 \circ M) = \coprod_E 1 \quad \longrightarrow \quad M \tag{10}$$

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

$$(\textstyle\coprod_E 1) \star F \cong \coprod_E (1 \star F) = \coprod_E (\int^{a,b} 1a \times Fb \times -^{a\times b}) \cong \coprod_E (\int^b Fb) \quad \to \quad F \tag{11}$$

Note that exception throwing for monads (10) and for applicatives (11) 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 3.5 (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 (9) for $\Sigma M = M \times M$ implies that

$$ph : M \times M, k : M \vdash \mu(\mathit{catch}\ ph, k) = \mathit{catch}\ \langle \mu(\pi_1\ ph,\ k), \mu(\pi_2\ ph,\ k)\rangle : M \tag{12}$$

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 $\langle \mathrm{Endo}(\mathscr{C}), \circ, \mathrm{Id} \rangle$, 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 3.4, and the second operation

$$\mathit{catch} : (\mathrm{Id} \times \mathrm{Id}) \circ M \circ M \cong (M \times M) \circ M \quad \longrightarrow \quad M \tag{13}$$

is catching. The trick here to avoid the undesirable equation (12) is that *catch* has after $M \times M$ an additional $- \circ M$ that represents an *explicit continuation* after the scoped operation *catch* [Piróg et al. 2018]: $\mathit{catch}\ (\langle p, h \rangle, k)$ is understood as handling the exception in p with h and then continuing as k. Then the generalised algebraicity (9) instantiates to

$$ph : M \times M, k : M, k' : M \vdash \mu(\mathit{catch}\ (ph, k), k') = \mathit{catch}\ (ph, \mu(k, k')) : M \tag{14}$$

Unlike (12), 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:

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

where $\eta : \mathsf{I} \to \tau$, $\mathit{throw} : 1 \to \tau$, and $\mathit{catch} : (\tau \times \tau) \sqcap \tau \to \tau$. These equations can be alternatively presented with an empty context by replacing all the k's with η like $\cdot \vdash \ \mathit{catch}(\langle \mathit{throw}, \eta \rangle, \eta) = \eta : \tau$ which is equivalent to the first equation above, since by (14), $\mathit{catch}(\langle x, y \rangle, \eta); k = \mathit{catch}(\langle x, y \rangle, k)$.

Example 3.6. Let S be a finite 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 $((\prod_S \mathsf{I}) \sqcap \tau) + ((\coprod_S \mathsf{I}) \sqcap \tau)$, whose first component represents an operation $g : (\prod_S \mathsf{I}) \sqcap \tau \to \tau$ reading the state, and the second component represents an operation $p : (\coprod_S \mathsf{I}) \sqcap \tau \to \tau$ 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

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

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].

3.3 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 3.7. 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 $T : \hat{\Sigma}'\text{-Alg} \to \hat{\Sigma}\text{-Alg}$ such that $\mathrm{U}_{\hat{\Sigma}} \circ T = \mathrm{U}_{\hat{\Sigma}'}$, where $\mathrm{U}_{\hat{\Sigma}} : \hat{\Sigma}\text{-Alg} \to \mathscr{C}$ and $\mathrm{U}_{\hat{\Sigma}'} : \hat{\Sigma}'\text{-Alg} \to \mathscr{C}$ are the forgetful functors. Equational systems on $\mathscr{C}$ and translations form a category $\mathrm{Eqs}(\mathscr{C})$, in which the identity arrows are the identity functors $\hat{\Sigma}\text{-Alg} \to \hat{\Sigma}\text{-Alg}$, and composition of translations $T \circ T'$ is functor composition.

Note the contravariance in the definition: a translation $\hat{\Sigma} \to \hat{\Sigma}'$ is a functor $\hat{\Sigma}'\text{-Alg} \to \hat{\Sigma}\text{-Alg}$ from the opposite direction. Informally, if one can translate operations Σ of $\hat{\Sigma}$ to terms $(\hat{\Sigma}')^*$ of $\hat{\Sigma}'$, then given an algebra $\Sigma' A \to A$ of the latter, it induces an algebra $\Sigma A \to (\Sigma')^* A \to A$ of the former.

Example 3.8. 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:

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

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

Colimits in $\mathrm{Eqs}(\mathscr{C})$ allows one to combine equational systems. The following are some elementary results about existence of colimits. Proofs or proof sketches of the results in this paper can be found in the supplemented appendices.

Lemma 3.9. *(i) When $\mathscr{C}$ has all (small) coproducts, so does the category $\mathrm{Eqs}(\mathscr{C})$. (ii) When $\mathscr{C}$ has binary coproducts and there is a reflection $\Sigma\text{-Alg} \rightleftarrows \hat{\Sigma}\text{-Alg}$, then $\mathrm{Eqs}(\mathscr{C})$ has coequalisers.*

Theorem 3.10. *The full subcategory $\mathrm{Eqs}_\omega(\mathscr{C}) \subseteq \mathrm{Eqs}(\mathscr{C})$ of equational systems whose functorial signature and context preserve colimits of ω-chains is cocomplete if $\mathscr{C}$ is cocomplete.*

Lastly, the following direct description of pushouts of inclusion translations is useful for combining theories of monoids extended with operations.

Lemma 3.11. *Let $\hat{\Sigma} \in \mathrm{Eqs}(\mathscr{C})$ 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 following is a pushout diagram of T_1 and T_2:*

$$\begin{array}{ccc} \hat{\Sigma} & \xrightarrow{\quad T_2 \quad} & \hat{\Sigma} \curvearrowleft_{\mathrm{op}} \Phi_2 \curvearrowleft_{\mathrm{eq}} E_2 \\ {\scriptstyle T_1}\downarrow & & \downarrow \\ \hat{\Sigma} \curvearrowleft_{\mathrm{op}} \Phi_1 \curvearrowleft_{\mathrm{eq}} E_1 & \longrightarrow & \hat{\Sigma} \curvearrowleft_{\mathrm{op}} (\Phi_1 + \Phi_2) \curvearrowleft_{\mathrm{eq}} E' \end{array}$$

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 $\alpha_i : (\Sigma + (\Phi_1 + \Phi_2))\text{-Alg} \to (\Sigma + \Phi_i)\text{-Alg}$ is the projection functor.

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 talk about their models and build free models (Theorem 3.2); (iii) and we can modularly *combine* such theories using colimits (§3.3). What remains to be done in the rest of the paper is developing *modular models*, which will allow us to combine models of existing theories into a model of a combined theory, thus achieving modularity for both syntax and semantics.

4 MONOIDAL THEORY FAMILIES

When defining modular models of theories, it is crucial to be clear about which theories are under consideration, since the more theories are considered, the more difficult it is to define a modular model for them. For example, if we only consider theories whose signatures are constant functors, then a model $\langle M, \alpha : A \to M \rangle$ and a model $\langle N, \beta : B \to N \rangle$ of constant signature functors K_A and K_B respectively (with no equations) can be easily combined into a model $\langle M+N, \alpha+\beta : A+B \to M+N \rangle$ of the combined signature $\mathrm{K}_A + \mathrm{K}_B$ using coproducts. But if the operation on N is of a more general form $\Sigma N \to N$, then there is no canonical way to construct a $\Sigma(M+N) \to M+N$. Thus in this section, we define an auxiliary notion of *monoidal theory families* before defining modular models, and we study some prominent examples and their connections.

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

Note that the coproducts in $\mathrm{Mon}/\mathrm{Eqs}(\mathscr{E})$ are equivalently pushouts $\hat{\Sigma} \leftarrow \mathrm{Mon} \to \hat{\Gamma}$ in $\mathrm{Eqs}(\mathscr{E})$, so coproducts in $\mathcal{F}$ are intuitively combining theories while identifying their monoid operations. This definition alone is boring, but its examples and their connections are interesting. The examples below assume a monoidal category with the following properties.

Definition 4.2. We call a monoidal category $\langle \mathscr{E}, \square, I \rangle$ *cocordial* when it is a **co**complete, with $\square : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ **co**continuous (i.e. $\square$ preserves colimits of ω-chains in both arguments), **r**ight **d**istributive (i.e. $\square$ preserves coproducts in the first argument) mono**id**a**l** category.

The monoidal categories introduced in §2.1, $\langle \mathrm{Endo}_f(\mathscr{C}), \circ, \mathrm{Id} \rangle$ for an lfp $\mathscr{C}$, $\langle \mathscr{C}, \times, 1 \rangle$ for a cocomplete cartesian $\mathscr{C}$, $\langle \mathrm{Endo}_f(\mathbf{Set}), \star, \mathrm{Id} \rangle$, and $\langle \mathrm{EndoPro}_s(\mathscr{C}), \otimes, I \rangle$ for a small $\mathscr{C}$, are all cocordial.

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

$$\{ \langle \Sigma\text{-Mon} \upharpoonleft_{\mathrm{eq}} E,\ T \rangle \mid A \in \mathscr{E},\ \Sigma = A \square -,\ E = (\mathrm{K}_B \vdash L = R) \} \tag{15}$$

where $T : \mathrm{Mon} \hookrightarrow \Sigma\text{-Mon} \upharpoonleft_{\mathrm{eq}} E$ is the inclusion translation, and E is an arbitrary functorial equation whose context is a *constant functor* (see the remark below). In other words, $\mathrm{ALG}(\mathscr{E})$ contains all equational systems extending the theory Σ-Mon (8) for some $\Sigma = A \square -$ (with the canonical pointed strength $(A \square X) \square Y \cong A \square (X \square Y)$) and also an equation.

The category $\mathrm{ALG}(\mathscr{E})$ satisfies the conditions in Definition 4.1 since it is closed under coproducts following Lemma 3.11, and all equational systems (15) in $\mathrm{ALG}(\mathscr{E})$ have free algebras by Theorem 3.2, since the signature and context functor of (15) preserve colimits of ω-chains.

In particular the theory of exceptions (Example 3.4) and states (Example 3.6) are in $\mathrm{ALG}(\mathscr{E})$. When $\mathscr{E} = \langle \mathrm{Endo}_f(\mathscr{C}), \circ, \mathrm{Id} \rangle$ over an lfp category $\mathscr{C}$, the family $\mathrm{ALG}(\mathscr{E})$ consists of theories of algebraic operations $A \circ M \to M$ for $A \in \mathrm{Endo}_f(\mathscr{C})$ on finitary monads M. When $\mathscr{E} = \langle \mathrm{Endo}_f(\mathbf{Set}), \star, \mathrm{Id} \rangle$, it then contains theories of applicatives F with 'applicative-algebraic' operations $A \star F \to F$.

Remark. The restriction in (15) that the functorial context must be a constant functor K_B needs some explanation: for a functor $\Sigma : \mathscr{E} \to \mathscr{E}$ with initial algebra Σ^* and a constant functor K_B, it is possible to show that arrows $B \to \Sigma^*$ in $\mathscr{E}$ are equivalently functorial terms $\Sigma\text{-Alg} \to \mathrm{K}_B\text{-Alg}$. Thus E in (15) is equivalently a pair of arrows $B \to (\Sigma + \Sigma_{\mathrm{Mon}})^*$, and indeed this is Kelly and Power [1993]'s way to specify equations.

The family $\mathrm{ALG}(\mathscr{E})$ has some interesting properties. The one below generalises the classical monad-theory correspondence [Linton 1966; Mahmoud 2010] to the generality of monoids.

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

Instantiating $\mathscr{E}$ by $\langle \mathrm{Endo}_f(\mathscr{C}), \circ, \mathrm{Id}\rangle$ for an lfp $\mathscr{C}$, we obtain the classical correspondence between finitary monads and (a slightly unconventional formulation of) first-order algebraic 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/arrows and the corresponding categories of theories of algebraic operations. This general monoid-theory correspondence seems new to us.

Another interesting property of $\mathrm{ALG}(\mathscr{E})$ is the following saying that almost all equational theories of operations on monoids can be turned into one in $\mathrm{ALG}(\mathscr{E})$ 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 $\mathrm{ALG}(\mathscr{E})$ is a coreflective subcategory of $\mathrm{Mon}/\mathrm{Eqs}_\omega(\mathscr{E})$, i.e. there is an adjunction $\mathrm{ALG}(\mathscr{E}) \underset{\top}{\overset{\lfloor - \rfloor}{\hookleftarrow}} \mathrm{Mon}/\mathrm{Eqs}_\omega(\mathscr{E})$. (ii) Moreover, the coreflector $\lfloor - \rfloor$ preserves initial algebras: for every $\langle \hat{\Sigma} \in \mathrm{Eqs}_\omega(\mathscr{E}), T : \mathrm{Mon} \to \hat{\Sigma}\rangle$, 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.*

Although $\mathrm{ALG}(\mathscr{E})$ 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 $\hat{\Sigma}\text{-Alg} \to \lfloor \langle \hat{\Sigma}, T\rangle \rfloor\text{-Alg}$, but these two categories of models are in general not equivalent.

Scoped Operations. Our next example of monoidal theory families is the family $\mathrm{SCP}(\mathscr{E})$ of *scoped (and algebraic) operations*, such as exception catching (Example 3.5). The family $\mathrm{SCP}(\mathscr{E})$ is given by the full subcategory of $\mathrm{Mon}/\mathrm{Eqs}(\mathscr{E})$ containing objects

$$\{\langle \Sigma\text{-Mon} \curvearrowleft_{\mathrm{eq}} E,\ T\rangle \mid A, B \in \mathscr{E},\ \Sigma = (A \square - \square -) + (B \square -),\ E = (\mathrm{K}_C \vdash L = R)\} \tag{16}$$

where $T : \mathrm{Mon} \to \Sigma\text{-Mon} \curvearrowleft_{\mathrm{eq}} E$ is the inclusion translation. The pointed strength of Σ needed in the definition of Σ-Mon (8) 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 3.5, 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 (9) 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 $\mathrm{ALG}(\mathscr{E}) \underset{\top}{\overset{\lfloor - \rfloor}{\hookleftarrow}} \mathrm{SCP}(\mathscr{E})$. 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 this example, we work concretely in the monoidal category $\langle \mathbf{Set}^{\mathbf{Fin}}, \bullet, V \rangle$ (3). A *binding signature* $\langle O, a \rangle$ consists of a set O of operations and an assignment $a : O \to \mathbb{N}^*$ of arity 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 $\Sigma_{\langle O,a \rangle} : \mathbf{Set}^{\mathbf{Fin}} \to \mathbf{Set}^{\mathbf{Fin}}$:

$$\Sigma_{\langle O,a \rangle} = \coprod_{o \in O,\ a(o) = \langle n_i \rangle_{1 \leqslant i \leqslant k}} \prod_{1 \leqslant i \leqslant k} (-)^{V^{n_i}} \tag{17}$$

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

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

where $T : \mathrm{Mon} \to \Sigma_{\langle O,a \rangle}\text{-Mon} \upharpoonleft_{\mathrm{eq}} E$ is the inclusion translation. The definition of $\mathrm{VAR}(\mathbf{Set}^{\mathbf{Fin}})$ satisfies Definition 4.1 because it is closed under coproducts by Lemma 3.11, and the functorial signature $\Sigma_{\langle O,a \rangle}$ and context 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 Ran_{V+1}, so $(-)^V$ preserves all colimits.

Again, the coreflector in Theorem 4.4 allows us to turn every theory in 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 initial algebra of $\lfloor \Lambda \rfloor$ is still λ-terms, but non-initial algebras of $\lfloor \Lambda \rfloor$ are quite different from those of Λ.

5 MODULAR MODELS OF MONOIDS

Now we are ready to show the main definition of this paper, *modular models* of a theory $\tilde{\Sigma}$ in a monoidal theory family $\mathcal{F}$. We have two equivalent formulations, one based on **CAT**-*valued functors*, and another based on *split fibrations*. The first one (Definition 5.2) is simpler to describe, while the second one (Prop. 5.5) is more convenient to work with. We also explain how modular models are used to interpret the abstract syntax of an equational system (§5.3).

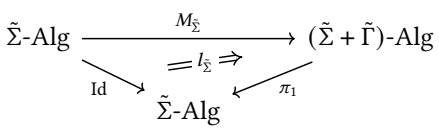
5.1 CAT-Valued-Functor-Based Formulation

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 $\mathrm{Mon} \to \hat{\Sigma}$. We abuse the notation $\tilde{\Sigma}$-Alg to mean the category $\hat{\Sigma}$-Alg of $\hat{\Sigma}$-algebras. In other words, we define a functor $(-)\text{-Alg} : \mathcal{F} \to \mathbf{CAT}$ to be $\mathcal{F} \xrightarrow{\pi_1} \mathrm{Eqs}(\mathscr{E}) \xrightarrow{(-)\text{-Alg}} \mathbf{CAT}$.

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 $M_{\tilde{\Sigma}} : \tilde{\Sigma}\text{-Alg} \to (\tilde{\Sigma} + \tilde{\Gamma})\text{-Alg}$ natural in $\tilde{\Sigma} \in \mathcal{F}$, i.e. M is a natural transformation

$$M : (-)\text{-Alg} \to (- + \tilde{\Gamma})\text{-Alg} : \mathcal{F} \to \mathbf{CAT}.$$

Also, a *lifting* l for a modular model M is a family $l_{\tilde{\Sigma}}$ of natural transformations for each $\tilde{\Sigma} \in \mathcal{F}$:



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

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

Example 5.3. For a trivial example, let $\mathcal{F}$ be the monoidal theory family containing only the theory Mon of monoids in $\mathscr{E}$ with the identity translation Mon $\to$ Mon. In this case, Mon-Alg is exactly the category Mon($\mathscr{E}$) of monoids in $\mathscr{E}$. A modular model M with a lifting l of $\langle \mathrm{Mon}, \mathrm{id} \rangle$ in $\mathcal{F}$ is precisely a (covariant) *monoid transformer* [Jaskelioff and Moggi 2010]: $\mathrm{Mon}(\mathscr{E}) \overset{\mathrm{Id}}{\underset{M}{\overset{\Downarrow l}{\rightrightarrows}}} \mathrm{Mon}(\mathscr{E})$. Thus monoid transformers (particularly, monad transformers) are special cases of modular models.

Definition 5.2 makes essential use of **CAT**-valued functors, and it is well known that **CAT**-valued functors are equivalent to *split fibrations* via the *Grothendieck construction*. Therefore we can alternatively formulate modular models based on fibrations. The alternative formulation is sometimes easier to work with, especially when thinking about liftings (whose definition in Definition 5.2 are in fact 3-morphisms in the 3-category of 2-categories) and certain 'dependently typed' constructions such as mapping each $\tilde{\Sigma}$ in $\mathcal{F}$ to the initial algebra in $\tilde{\Sigma}$-Alg.

5.2 Fibration-Based Formulation

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 appendices.

It is well known that **CAT**-valued functors are equivalent to split fibrations by the *Grothendieck construction*. In particular, for every monoidal theory family $\mathcal{F}$, the **CAT**-valued functor $(-)$-Alg : $\mathcal{F} \to \mathbf{CAT}$ induces a category $\mathcal{F}$-Alg and a split fibration $\mathcal{F}$-Alg $\to \mathcal{F}$, which we explicitly describe below. The intuition is that $\mathcal{F}$-Alg is the category of *all models of all theories* in $\mathcal{F}$.

Definition 5.4. For every monoidal theory family $\mathcal{F}$, the objects of category $\mathcal{F}$-Alg are tuples

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

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 $T : \hat{\Sigma} \to \hat{\Gamma}$ is a functorial translation in $\mathcal{F}$, and the other component $h : A \to B \in \mathscr{E}$ is a $\hat{\Sigma}$-algebra homomorphism from $\langle A, \alpha \rangle$ to $T\langle B, \beta \rangle$:

$$\begin{array}{ccc} \Sigma A & \xrightarrow{\alpha} & A \\ {\scriptstyle \Sigma h}\downarrow & & \downarrow \boxed{h} \\ \Sigma B & \xrightarrow[T\langle B,\beta\rangle]{} & B \end{array} \quad (\text{in } \hat{\Sigma}\text{-Alg}) \qquad \xleftarrow{\boxed{T}} \qquad \Gamma B \xrightarrow{\beta} B \quad (\text{in } \hat{\Gamma}\text{-Alg})$$

The identities arrows are pairs of identity translations and homomorphisms: $\langle \mathrm{Id} : \hat{\Sigma} \to \hat{\Sigma}, \mathrm{id} : A \to A \rangle$. 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 $\to \mathcal{F}$ 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 $T : \langle \hat{\Sigma}, T_\Sigma \rangle \to \langle \hat{\Gamma}, T_\Gamma \rangle \in \mathcal{F}$ and objects $\langle \hat{\Gamma}, T_\Gamma, B, \beta \rangle \in \mathcal{F}$-Alg to $\langle T, \mathrm{id} \rangle : \langle \hat{\Sigma}, T_\Sigma, B, T\langle B, \beta \rangle \rangle \to \langle \hat{\Gamma}, T_\Gamma, B, \beta \rangle$.

Given a theory $\tilde{\Gamma} \in \mathcal{F}$, we are also interested in models of theories 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 $\to \mathcal{F}$ along the functor $(- + \tilde{\Gamma}) : \mathcal{F} \to \mathcal{F}$, which is the following pullback in

the category **CAT** of categories:

$$\begin{array}{ccc} (\mathcal{F}+\tilde{\Gamma})\text{-Alg} & \xrightarrow{\mathrm{K}} & \mathcal{F}\text{-Alg} \\ {\scriptstyle Q}\downarrow & \lrcorner & \downarrow{\scriptstyle P} \\ \mathcal{F} & \xrightarrow[-+\tilde{\Gamma}]{} & \mathcal{F} \end{array} \tag{18}$$

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

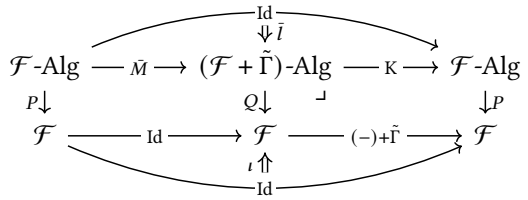
$$\langle \hat{\Sigma} \in \mathrm{Eqs}(\mathscr{E}),\quad T_\Sigma : \mathrm{Mon} \to \hat{\Sigma},\quad A \in \mathscr{E},\quad \alpha : \Sigma A \to A,\quad \beta : \Gamma A \to A \rangle$$

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 $T_\Gamma \langle A, \alpha \rangle = T_\Sigma \langle A, \beta \rangle \in \mathrm{Mon}(\mathscr{E})$. 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 (18) also induces two functors Q and K. The functor $Q : (\mathcal{F}+\tilde{\Gamma})\text{-Mon} \to \mathcal{F}$ is the projection to $\mathcal{F}$, and it is a split fibration with a split cleavage similar to that of P. The functor $\mathrm{K} : (\mathcal{F}+\hat{\Gamma})\text{-Mon} \to \mathcal{F}\text{-Mon}$ 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 $\langle \mathrm{K}, - + \hat{\Gamma} \rangle$ is a split fibration morphism from Q to P.

Now we have enough machinery to spell out the fibration-based formulation of modular models.

Proposition 5.5. *A modular model of some $\tilde{\Gamma} \in \mathcal{F}$ as in Definition 5.2 is equivalently a functor $\bar{M} : \mathcal{F}\text{-Alg} \to (\mathcal{F}+\tilde{\Gamma})\text{-Alg}$ such that $\langle \bar{M}, \mathrm{Id} \rangle$ is a split fibration morphism from P to Q as below. And a lifting for $\bar{M}$ is equivalently a natural transformation $\bar{l} : \mathrm{Id} \to K \circ \bar{M}$ such that $\langle \bar{l}, \iota \rangle$ is a split fibration 2-cell $\langle \mathrm{Id}, \mathrm{Id} \rangle \to \langle K \circ \bar{M}, - + \tilde{\Gamma} \rangle$ where $\iota : \mathrm{Id} \to - + \tilde{\Gamma}$ is the coprojection:*



Compared to Definition 5.2, liftings in the fibration-based formulation are natural transformations instead of morphisms between natural transformations. Also, the 'dependently typed' mapping sending every $\tilde{\Sigma} = \langle \hat{\Sigma}, T_\Sigma \rangle \in \mathcal{F}$ to its initial algebra $\hat{\Sigma}^*$ in $\hat{\Sigma}$-Alg now can be conveniently formulated as a functor $(-)^\star : \mathcal{F} \to \mathcal{F}$-Alg defined by

$$\tilde{\Sigma} \mapsto \langle \hat{\Sigma},\ T_\Sigma,\ \hat{\Sigma}^*, \alpha^\Sigma : \Sigma(\hat{\Sigma}^*) \to \hat{\Sigma}^* \rangle \quad (T : \tilde{\Sigma} \to \tilde{\Gamma}) \mapsto \langle T,\ u : \langle \hat{\Sigma}^*,\ \alpha^\Sigma \rangle \to T\langle \hat{\Gamma}^*,\ \alpha^\Gamma \rangle \rangle \tag{19}$$

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

5.3 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 $\hat{\Gamma}^*$, then for every ordinary model $\langle A, \alpha \rangle$ of $\hat{\Gamma}$, the unique $\hat{\Gamma}$-homomorphism from $\hat{\Gamma}^*$ 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 $\langle \hat{\Sigma}^*, \alpha^\Sigma \rangle$, which is mapped by $M_{\tilde{\Sigma}} : \tilde{\Sigma}\text{-Alg} \to (\tilde{\Sigma}+\tilde{\Gamma})\text{-Alg}$ to an algebra of $\tilde{\Sigma}+\tilde{\Gamma}$. Then the initial algebra $(\tilde{\Sigma}+\tilde{\Gamma})^*$ of (the equational system part of) $\tilde{\Sigma}+\tilde{\Gamma}$ induces a unique homomorphism:

$$\tilde{h}_{\tilde{\Sigma}} : (\tilde{\Sigma}+\tilde{\Gamma})^* \to M_{\tilde{\Sigma}} \langle \hat{\Sigma}^*, \alpha^\Sigma \rangle. \tag{20}$$

The intuition is that this morphism *modularly* interprets the $\hat{\Gamma}$-operations in the abstract syntax $(\tilde{\Sigma} + \tilde{\Gamma})^*$ with a modular model M, leaving operations from the other theory $\tilde{\Sigma}$ uninterpreted.

Specially, let $\tilde{\Sigma}$ be the theory $\langle \mathrm{Mon}, \mathrm{id} \rangle$ 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 $\hat{\Sigma}^*$ is the initial monoid I and $(\tilde{\Sigma} + \tilde{\Gamma})^* \cong \hat{\Gamma}^*$. In this case, the morphism $\tilde{h}_{\tilde{\Sigma}} : \hat{\Gamma}^* \to \bar{M}\langle I, \alpha^I \rangle$ (20) interprets the abstract syntax $\hat{\Gamma}^*$ without anymore uninterpreted operations.

The interpretation (20) can be formulated as a natural transformation in $\tilde{\Sigma}$ by using the functor $(-)^\star : \mathcal{F} \to \mathcal{F}$-Alg (19) sending every $\tilde{\Sigma} \in \mathcal{F}$ to its initial algebra in $\mathcal{F}$-Alg.

Proposition 5.6. *Given a modular model M of $\tilde{\Gamma} \in \mathcal{F}$, there is a natural transformation*

$$h^M : (- + \tilde{\Gamma})^\star \to K\bar{M}(-)^\star : \mathcal{F} \to \mathcal{F}\text{-Alg} \tag{21}$$

such that all $h^M_{\tilde{\Sigma}} = \langle \mathrm{id}_{\tilde{\Sigma}+\tilde{\Gamma}}, \tilde{h}_{\tilde{\Sigma}} \rangle$, where $\bar{M} : \mathcal{F}\text{-Alg} \to (\mathcal{F} + \tilde{\Gamma})\text{-Alg}$ is the equivalent form of M in Prop. 5.5, and $\mathrm{K} : (\mathcal{F} + \tilde{\Gamma})\text{-Alg} \to \mathcal{F}\text{-Alg}$ is defined as in (18).

Remark. Since $\tilde{\Sigma} + \tilde{\Gamma}$ extends the theory of monoids, the interpretation (21) 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 5.2) are useful for *reusing existing interpretations*. Suppose that some operations are first modelled by a theory $\tilde{\Sigma}$ and interpreted with some model $A \in \tilde{\Sigma}$-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 $\hat{\Sigma}^*$, the following diagram commutes:

$$\begin{array}{lcc} \text{Syntax} & \hat{\Sigma}^* \xrightarrow{\iota_1^*} (\tilde{\Sigma} + \tilde{\Gamma})^* & \\ & {\scriptstyle u_1}\downarrow \qquad\qquad \downarrow{\scriptstyle u_2} & \text{in } \tilde{\Sigma}\text{-Alg} \\ \text{Semantics} & A \xrightarrow{l_A} M_{\tilde{\Sigma}} A & \end{array}$$

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

6 CONSTRUCTIONS AND EXAMPLES OF MODULAR MODELS

Now we show general constructions and concrete examples of modular models. Our constructions can be roughly divided into three kinds: constructing from *monoid transformers*, *composition* of modular models, and constructions in symmetric monoidal categories.

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 a more elaborate version 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 usually be upgraded into modular models, and there are two general results: one for theories in $\mathrm{ALG}(\mathscr{E})$ from monoid transformers (Theorem 6.2) and ordinary models (Corollary 6.3), another for theories in $\mathrm{SCP}(\mathscr{E})$ from *functorial monoid transformers* (Theorem 6.5).

Example 6.1. Let us start with a concrete example. Let $\mathscr{E}$ be $\langle \mathrm{Endo}_f(\mathscr{C}), \circ, \mathrm{Id} \rangle$ for lfp $\mathscr{C}$. A modular model M for the theory ET_E of *exception throwing* (Example 3.4) in the family $\mathrm{ALG}(\mathscr{E})$ of

algebraic operations is given by the following family of functors $M_{\tilde{\Sigma}} : \tilde{\Sigma}\text{-Alg} \to (\tilde{\Sigma} + \mathrm{ET}_E)\text{-Alg}$ for all $\tilde{\Sigma} \in \mathrm{ALG}(\mathscr{E})$. Recall that objects of $\tilde{\Sigma}$-Alg are tuples as follows satisfying certain equations:

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

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 $\mathrm{Endo}_f(\mathscr{C})$. The carrier is equipped with operations $[\alpha^\sharp, \beta] :\ (\Sigma \circ C_A) + \mathbb{E} \to C_A$, 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 $d : (\mathbb{E} + \mathrm{Id}) \circ A \to A \circ (\mathbb{E} + \mathrm{Id})$ 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 $h : A \to B$ to $h \circ (\mathbb{E} + I) : C_A \to C_B$. The modular model M has a lifting $l_{\tilde{\Sigma}, \langle A,\ \alpha \rangle} = [\![a : A \vdash (a,\ \mathsf{inj}_2\ *) : C_A]\!]$.

With this modular model, we can interpret terms $(\tilde{\Sigma} + \tilde{\Gamma})^*$ of exception throwing mixed with other algebraic operations $\tilde{\Sigma} \in \mathrm{ALG}(\mathscr{E})$: (20) specialises to $h_{\tilde{\Sigma}} : (\tilde{\Sigma} + \tilde{\Gamma})^* \to (\hat{\Sigma}^* \circ (\mathbb{E} + \mathrm{Id}))$.

The main ingredient in the last example is the exception monad transformer. This construction can be done in a general way: every covariant monoid transformer that are equipped with a $\hat{\Gamma}$-algebra can be turned into a modular model of $\hat{\Gamma}$ in $\mathrm{ALG}(\mathscr{E})$.

Theorem 6.2. *For each $\tilde{\Gamma} = \langle \hat{\Gamma}, T_\Gamma \rangle \in \mathrm{ALG}(\mathscr{E})$ over a cocordial monoidal category $\mathscr{E}$, a functor $H : \mathrm{Mon}(\mathscr{E}) \to \hat{\Gamma}\text{-Alg}$ and a $\tau : \mathrm{Id} \to T_\Gamma \circ H$ can be extended to a modular model $\bar{M}$ of $\hat{\Gamma}$ such that the diagram on the right below commutes, and $\bar{M}$ has a lifting $\bar{l}_{\langle \langle \hat{\Sigma}, T_\Sigma \rangle,\ A,\ \alpha \rangle} = \tau_{\langle A,\ T_\Sigma \alpha \rangle}$:*

$$\begin{array}{ccc} & \hat{\Gamma}\text{-Alg} & \\ {\scriptstyle H}\nearrow & \Uparrow{\scriptstyle \tau} & \searrow{\scriptstyle T_\Gamma} \\ \mathrm{Mon}(\mathscr{E}) & \xrightarrow[\mathrm{Id}]{} & \mathrm{Mon}(\mathscr{E}) \end{array} \qquad \begin{array}{ccc} \mathcal{F}\text{-Alg} & \xrightarrow{\bar{M}} & (\mathcal{F} + \tilde{\Gamma})\text{-Alg} \\ \downarrow & & \downarrow \\ \mathrm{Mon}(\mathscr{E}) & \xrightarrow[H]{} & \hat{\Gamma}\text{-Alg} \end{array} \tag{22}$$

where the unlabelled vertical arrows are the evident projection functors.

This theorem has a wide range of applications. The *state monad transformer* $A \mapsto (A(S \times -))^S$ for a finite set S together with its model for the theory St_S of *mutable state* (Example 3.6) yields a modular model of St_S in $\mathrm{ALG}(\mathrm{Endo}_f(\mathscr{C}))$. 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 $\mathrm{ALG}(\mathscr{E})$ from ordinary models by taking coproducts of monoids.

[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.

Corollary 6.3. Let $\hat{A} = \langle A, \alpha \rangle \in \hat{\Gamma}\text{-Alg}$ be an ordinary model of $\hat{\Gamma}$-Alg for $\tilde{\Gamma} = \langle \hat{\Gamma}, T_\Gamma \rangle \in \mathrm{ALG}(\mathscr{E})$. 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 $H : \mathrm{Mon}(\mathscr{E}) \to \hat{\Gamma}\text{-Alg}$ mapping every monoid $\hat{M}$ to the $\hat{\Gamma}$-algebra $\hat{M} +_{\mathrm{Mon}} (T_\Gamma \hat{A})$ equipped with

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

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

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

Example 6.4. The theory EC of *exception throwing and catching* in Example 3.5 is in the family $\mathrm{SCP}(\mathscr{E})$ for $\mathscr{E} = \langle \mathrm{Endo}_f(\mathbf{Set}), \circ, \mathrm{Id} \rangle$. A modular model for it can be constructed by extending the modular model of throwing in Example 6.1 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 $\mathit{catch} : (C_A \times C_A) \circ C_A \to C_A$, we denote by s the following canonical strength in $\mathrm{Endo}_f(\mathbf{Set})$:

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

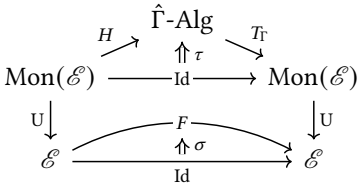
Then *catch* is $s \circ C_A$ followed by the arrow denoted by the following term:

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

(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 6.2, modular models for theories $\hat{\Gamma} \in \mathrm{SCP}(\mathscr{E})$ 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 6.5. *For each $\langle \hat{\Gamma}, T_\Gamma \rangle \in \mathrm{SCP}(\mathscr{E})$ over a cocordial and* closed *monoidal category $\mathscr{E}$, a functor $H : \mathrm{Mon}(\mathscr{E}) \to \hat{\Gamma}\text{-Alg}$ 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 $\mathrm{U} \circ T_\Gamma \circ H = F \circ \mathrm{U}$ and $\tau \circ \mathrm{U} = \sigma \circ U$ can be extended to a modular model M of $\langle \hat{\Gamma}, T_\Gamma \rangle$ with a lifting l such that (22) commutes and $l_{\langle \langle \hat{\Sigma}, T_\Sigma \rangle,\ A,\ \alpha \rangle} = \tau_{\langle A,\ T_\Sigma \alpha \rangle}$.*



In comparison to Theorem 6.2, the theorem above requires a closed $\mathscr{E}$ and the monoid transformer $\langle T_\Gamma \circ H, \tau \rangle$ to be *functorial*, i.e. being 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 6.4, the state monad transformer $A \mapsto (A(S \times -))^S$, and the free monad transformer (also known as the resumption monad transformer) $A \mapsto \mu X.\, A(- + SX)$ for finitary endofunctors $S : \mathscr{C} \to \mathscr{C}$ [Cenciarelli and Moggi 1993].

6.2 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 6.6. Given a modular model $\bar{M}$ of $\tilde{\Gamma} \in \mathcal{F}$ and another $\bar{N}$ of $\tilde{\Phi} \in \mathcal{F}$ in the form of Prop. 5.5, their *composite* $\bar{M} \triangleright \bar{N}$ is a modular model of $\tilde{\Gamma} + \tilde{\Phi}$:

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

defined by the pullback property (18) for $(\mathcal{F} + \tilde{\Gamma} + \tilde{\Phi})$-Alg and the commutativity of the diagram:

$$\begin{array}{ccccccccc}
\mathcal{F}\text{-Alg} & \xrightarrow{\bar{M}} & (\mathcal{F}+\tilde{\Gamma})\text{-Alg} & \xrightarrow{\mathrm{K}} & \mathcal{F}\text{-Alg} & \xrightarrow{\bar{N}} & (\mathcal{F}+\tilde{\Phi})\text{-Alg} & \xrightarrow{\mathrm{K}} & \mathcal{F}\text{-Alg} \\
{\scriptstyle P}\downarrow & & {\scriptstyle Q}\downarrow & & {\scriptstyle P}\downarrow & & {\scriptstyle Q'}\downarrow & & {\scriptstyle P}\downarrow \\
\mathcal{F} & \xrightarrow[\mathrm{Id}]{} & \mathcal{F} & \xrightarrow[-+\tilde{\Gamma}]{} & \mathcal{F} & \xrightarrow[\mathrm{Id}]{} & \mathcal{F} & \xrightarrow[-+\tilde{\Phi}]{} & \mathcal{F}
\end{array}$$

Also, liftings $\bar{l}^N$ of N and $\bar{l}^M$ of M in the form of Prop. 5.5 compose a lifting $\bar{l}^N \circ \bar{l}^M$ of $\bar{M} \triangleright \bar{N}$.

Example 6.7. Let M_E be the modular model of *exception* throwing and catching (Example 6.4), and M_S be the modular model of *mutable state* arising from the state monad transformer [Liang et al. 1995] by Theorem 6.2. 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 6.7 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 (21). 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 $(\hat{\Sigma} + (\hat{\Gamma} + \hat{\Phi}))^\star$.

Lemma 6.8 (Fusion). *Given modular models $\hat{M}$ of $\hat{\Gamma}$ and $\hat{N}$ of $\hat{\Phi}$, the following diagram commutes:*

$$\begin{array}{ccc}
(\tilde{\Sigma} + (\tilde{\Gamma} + \tilde{\Phi}))^\star & \xrightarrow{\cong} & ((\tilde{\Sigma} + \tilde{\Gamma}) + \tilde{\Phi})^\star \\
{\scriptstyle h^{M \triangleright N}_{\tilde{\Sigma}}}\downarrow & & \downarrow{\scriptstyle h^{N}_{\tilde{\Sigma}+\tilde{\Gamma}}} \\
\mathrm{K}N\mathrm{K}M\tilde{\Sigma}^\star & \xleftarrow{KNh^{M}_{\tilde{\Sigma}}} & \mathrm{K}N(\tilde{\Sigma} + \tilde{\Gamma})^\star
\end{array}$$

This result is an instance of *short-cut fusion* [Ghani and Johann 2007; Gill et al. 1993] and generalises the fusion of algebraic effect handlers by Yang and Wu [2021]. 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.

6.3 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 $\langle \mathrm{Endo}_f(\mathbf{Set}), \star, \mathrm{Id} \rangle$ 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 $A \square B$:

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

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

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

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

Theorem 6.9 (Independent Combination)**.** *Let $\mathscr{E}$ be a symmetric cocordial monoidal category and $\mathcal{F}$ be* ALG($\mathscr{E}$) *or* SCP($\mathscr{E}$)*. For each $\tilde{\Gamma} \in \mathcal{F}$, every ordinary model $\hat{A} \in \tilde{\Gamma}$-Alg induces a modular model M of $\tilde{\Gamma}$ such that $M_{\tilde{\Sigma}}\langle B, \beta\rangle$ is carried by $A \square B$, and M has a lifting $l_{\tilde{\Sigma},\langle B,\beta\rangle} = [\![b : B \vdash (\eta^A,\ b) : A \square B]\!]$.*

For $\mathscr{E} = \langle \mathrm{Endo}_f(\mathbf{Set}), \star, \mathrm{Id}\rangle$, 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 6.10 (Dependent Combination)**.** *Let $\mathscr{E}$ be a $\langle \mathrm{Endo}_f\mathbf{Set}, \star, \mathrm{Id}\rangle$ and $\mathcal{F}$ be* ALG($\mathscr{E}$) *or* SCP($\mathscr{E}$)*. For each $\tilde{\Gamma} \in \mathcal{F}$, every ordinary model $\hat{A} \in \tilde{\Gamma}$-Alg induces a 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 6.11. To highlight the difference between Theorem 6.9 and Theorem 6.10, 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 ET_E of exception *throwing*, equipped with an operation $throw : \mathbb{E} \star (\mathbb{E} + \mathrm{Id}) \longrightarrow (\mathbb{E} + \mathrm{Id})$. Using Theorem 6.10, 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 $x, y \in (\mathbb{E} + B)X$, $throw\ \langle e, x\rangle = inj_1\ e = throw\ \langle e, y\rangle$, which means that exception throwing discards any B-computation. But it is not true for the independent composition $(\mathbb{E} + \mathrm{Id}) \star B$ using Theorem 6.9.

Staging. As our final example, we show an interesting modular model for *staging*, generalising a construction that Kidney and Wu [2021] introduce for *breadth-first search* with applicative functors. The theory $\mathrm{Stg} \in \mathrm{SCP}(\mathscr{E})$ has a unary scoped operation $later : A \square A \cong I \square A \square A \longrightarrow A$. The intuition is that a staged program has multiple stages of execution, and the operation $later\ \langle p, k\rangle$ delays the execution of p by a stage, and continues as k. For example, the program

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

is supposed to execute p_{11} and p_{12} at stage 1, p_{22} and p_{21} at stage 2, and p_3 at stage 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 $S_A = \mu X.\ A \square X + I$ over A with a nonstandard monoid structure $\langle S_A, \mu^{S_A}, \eta^{S_A}\rangle$ with $\eta^{S_A} = [\![\cdot \vdash out^\circ\ (\mathrm{inj}_2 *)]\!]$ and μ^{S_A} being

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

where $out : (S_A \cong A \square S_A + I) : out^\circ$ 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 $S_A = \mu X.A \square X + I$ is a list of A-computations at each stage, and μ^{S_A} merges two lists stage-by-stage. The *later* operation on S_A is defined as $[\![(p : S_A, k : S_A \vdash \mu^{S_A}(out^\circ(\mathrm{inj}_1(\eta^A, p), k)) : S_A]\!]$, and it turns out algebraic and scoped operations on A can be lifted to S_A. In summary, we have the following:

Proposition 6.12. *Let $\mathscr{E}$ be a symmetric closed cocordial monoidal category. There is a modular model M of the theory* Stg *of staging in* SCP($\mathscr{E}$)*, such that $M_{\tilde{\Sigma}}\langle A, \alpha\rangle$ is carried by $\mu X.\ A \square X + I$.*

7 RELATED WORK AND CONCLUSION

Effect Handlers. The most closely related work is the line of research on *handlers of algebraic effects* introduced by Plotkin and Pretnar [2009, 2013]. Semantically, handlers are models of first-order algebraic theories, and are used for interpreting free algebras. As a programming construct,

handlers offer a composable approach to user-defined algebraic effects, essentially relying on the fact that algebraic operations can be lifted canonically. Many implementations of handlers have been developed, both as libraries (e.g. Kammar et al. [2013]) and languages (e.g. Bauer and Pretnar [2015]). In particular, our work is inspired by Schrijvers et al. [2019]'s *modular handlers* in Haskell, which are handlers polymorphic in the unhandled effects.

Handlers of algebraic effects have been generalised in several directions. Wu et al. [2014] observe that implementing *scoped operations* as handlers leads to non-modularity issues, and they propose modelling these operations with higher-order abstract syntax and generalising handlers to this setting. Later, the categorical foundation of handlers of scoped operations were studied by Piróg et al. [2018] and Yang et al. [2022]. In another direction, Pieters et al. [2020] generalised handlers from algebraic operations on monads to monoids. In comparison to the present work, Piróg et al. [2018] and Yang et al. [2022] do not consider modularity of models, and Pieters et al. [2020] only consider algebraic operations. Hence our initial motivation of the present work was to develop a clear categorical formulation of equational theories of not necessarily algebraic operations and their modular models in the generality of monoids.

A notable deviation of our definition of modular models from the standard notion of effect handlers is that our modular models are required to have a monoid structure, rather than only modelling the operations. The distinction is analogous to Arkor and Fiore [2020]'s models of *simply typed syntax* versus *simple type theories*. We believe that our deviation is a reasonable choice since (i) many practical examples of modular handler have a monoid structure anyway; and (ii) many theories of non-algebraic operations inherently involve monoid operations in their equational laws, such as Example 3.5, so a handler for such a theory without a monoid structure makes little sense.

Monad Transformers. Moggi [1989a,b], Wadler [1990], and Spivey [1990] pioneered using *monads* to model computational effects in programming languages. To achieve modularity, Moggi [1989a,b] introduced *monad transformers*. Later, Jaskelioff [2009] showed how to lift $\hat{\Sigma}$-operations along *functorial monad transformers*, and this result was generalised to *monoid transformers* by Jaskelioff and Moggi [2010]. Our notion of *modular models* is a conceptual development of monoid transformers: besides transforming monoids into new ones, how to equip them with new operations and lift existing operations are also part of the definition. Thus philosophically, modular models are more like Plotkin and Pretnar [2013]'s handlers, since they are meant to interpret initial algebras of equational theories that model computational effects, rather than modelling the effects themselves.

Theories and Syntax. *Algebraic theories* and their connections to *Lawvere theories* and *monads* have been studied for decades (see e.g. [Adamek et al. 2010]). In this paper, we use Fiore and Hur [2009]'s *equational systems* to define equational theories for their generality and simplicity. *Abstract syntax* can be modelled as initial algebras of theories [Goguen et al. 1977], and Fiore et al. [1999] show that abstract syntax with variable bindings can be modelled as initial algebras in a presheaf category, and they introduce Σ*-monoids*. Building on their work, we have investigated connections between families of theories of Σ-monoids, and their modular models.

Conclusion. We have developed a categorical framework of modular models of equational theories. We applied this framework to families of algebraic and scoped operations on monoids, bridging equational theories and monoid transformers. As future work, we can 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

Jiri Adámek, Stefan Milius, Nathan Bowler, and Paul B. Levy. 2012. Coproducts of Monads on Set. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science* (New Orleans, Louisiana) *(LICS '12)*. IEEE Computer Society, USA, 45–54. https://doi.org/10.1109/LICS.2012.16

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

J. Adamek, J. Rosicky, E. M. Vitale, and F. W. Lawvere. 2010. *Algebraic Theories*. Cambridge University Press, Cambridge. 413–418 pages. https://doi.org/10.1017/CBO9780511760754

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

Nathanael Arkor and Marcelo Fiore. 2020. Algebraic Models of Simple Type Theories: A Polynomial Approach. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science* (Saarbrücken, Germany) *(LICS '20)*. Association for Computing Machinery, New York, NY, USA, 88–101. https://doi.org/10.1145/3373718.3394771

Andrej Bauer and Matija Pretnar. 2015. Programming with algebraic effects and handlers. *Journal of Logical and Algebraic Methods in Programming* 84, 1 (2015), 108–123. https://doi.org/10.1016/j.jlamp.2014.02.001

Richard Bird and Oege de Moor. 1997. *Algebra of Programming*. Prentice-Hall, Inc., USA.

Robert Cartwright and Matthias Felleisen. 1994. Extensible Denotational Language Specifications. In *SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SOFTWARE, NUMBER 789 IN LNCS*. Springer-Verlag, 244–272.

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

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. 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 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).

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

J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. 1977. Initial Algebra Semantics and Continuous Algebras. *J. ACM* 24, 1 (Jan. 1977), 68–95. https://doi.org/10.1145/321992.321997

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.

John Hughes. 2000. Generalising monads to arrows. *Science of Computer Programming* 37, 1 (2000), 67–111. https://doi.org/10.1016/S0167-6423(99)00023-4

Martin Hyland, Gordon Plotkin, and John Power. 2006. Combining Effects: Sum and Tensor. *Theor. Comput. Sci.* 357, 1 (July 2006), 70–99. https://doi.org/10.1016/j.tcs.2006.03.013

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

Bart Jacobs, Chris Heunen, and Ichiro Hasuo. 2009. Categorical semantics for arrows. *Journal of Functional Programming* 19, 3-4 (2009), 403–438. https://doi.org/10.1017/S0956796809007308

Mauro Jaskelioff. 2009. Modular Monad Transformers. In *Programming Languages and Systems*, Giuseppe Castagna (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 64–79. https://doi.org/10.1007/978-3-642-00590-9_6

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

Ohad Kammar, Sam Lindley, and Nicolas Oury. 2013. Handlers in Action. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming* (Boston, Massachusetts, USA) *(ICFP '13)*. Association for Computing Machinery, New York, NY, USA, 145–158. https://doi.org/10.1145/2500365.2500590

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

G. M. Kelly. 1982. Structures defined by finite limits in the enriched context, I. *Cahiers de Topologie et Géométrie Différentielle Catégoriques* 23, 1 (1982), 3–42.

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

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

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

Sam Lindley, Philip Wadler, and Jeremy Yallop. 2011. Idioms are Oblivious, Arrows are Meticulous, Monads are Promiscuous. *Electronic Notes in Theoretical Computer Science* 229, 5 (2011), 97–117. https://doi.org/10.1016/j.entcs.2011.02.018 Proceedings of the Second Workshop on Mathematically Structured Functional Programming (MSFP 2008).

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.

Ola Mahmoud. 2010. *Second-Order Algebraic Theories.* Ph.D. Dissertation. University of Cambridge. https://doi.org/10.17863/CAM.16369

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

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

Ruben P. Pieters, Exequiel Rivas, and Tom Schrijvers. 2020. Generalized monoidal effects and handlers. *Journal of Functional Programming* 30 (2020), e23. https://doi.org/10.1017/S0956796820000106

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. 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. 2009. Handlers of Algebraic Effects. In *Programming Languages and Systems*, Giuseppe Castagna (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 80–94. https://doi.org/10.1007/978-3-642-00590-9_7

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

Tom Schrijvers, Maciej Piróg, Nicolas Wu, and Mauro Jaskelioff. 2019. Monad Transformers and Modular Algebraic Effects: What Binds Them Together. In *Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell* (Berlin, Germany) *(Haskell 2019)*. Association for Computing Machinery, New York, NY, USA, 98–113. https://doi.org/10.1145/3331545.3342595

Mike Spivey. 1990. A functional theory of exceptions. *Science of Computer Programming* 14, 1 (1990), 25–42. https://doi.org/10.1016/0167-6423(90)90056-J

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.

Philip Wadler. 1990. Comprehending Monads. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming* (Nice, France) *(LFP '90)*. ACM, 61–78. https://doi.org/10.1145/91556.91592

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

Philip Wadler. 1998. The Expression Problem. https://homepages.inf.ed.ac.uk/wadler/papers/expression/expression.txt Online; accessed 11-October-2021.

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

Zhixuan Yang and Nicolas Wu. 2021. Reasoning about Effect Interaction by Fusion. *Proc. ACM Program. Lang.* 5, ICFP, Article 73 (Aug. 2021), 29 pages. https://doi.org/10.1145/3473578

A BASICS OF FIBRATIONS

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

$$\begin{array}{ccc} PZ & & \\ {\scriptstyle w}\downarrow & \searrow^{Pg} & \\ PX & \xrightarrow[Pf]{} & PY \end{array} \quad \text{in } \mathscr{B} \qquad \Longrightarrow \qquad \begin{array}{ccc} Z & & \\ {\scriptstyle h}\vdots & \searrow^{g} & \\ X & \xrightarrow[f]{} & Y \end{array} \quad \text{in } \mathscr{T}$$

A *split fibration over* $\mathscr{B}$ is a functor $P : \mathscr{T} \to \mathscr{B}$ equipped with a mapping κ, called a *split cleavage*, sending every pair $\langle Y, u \rangle$ of an arrow $u : I \to J \in \mathscr{B}$ and an object $Y \in \mathscr{T}$ with $PY = J$ to a cartesian morphism $\kappa(Y, u) : X \to Y$ 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 (23) commute and preserving the cleavage $F\kappa(Y, u) = \kappa(FY, Gu)$:

$$\begin{array}{ccc} \mathscr{T} & \xrightarrow{F} & \mathscr{T}' \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P'} \\ \mathscr{B} & \xrightarrow[G]{} & \mathscr{B}' \end{array} \qquad \begin{array}{ccc} \mathscr{T} & \overset{F}{\underset{F'}{\Downarrow \sigma}} & \mathscr{T}' \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P'} \\ \mathscr{B} & \underset{G}{\overset{G'}{\Uparrow \tau}} & \mathscr{B}' \end{array} \tag{23}$$

Split fibrations and morphisms form a category 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 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 (23).

Definition A.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 $\mathrm{Fib}_s(\mathscr{B})$ 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.

B PROOF SKETCHES

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

Lemma 3.9. *(i) When $\mathscr{C}$ has all (small) coproducts, so does the category* $\mathrm{Eqs}(\mathscr{C})$*. (ii) When $\mathscr{C}$ has binary coproducts and there is a reflection* $\Sigma\text{-Alg} \underset{\perp}{\rightleftarrows} \hat{\Sigma}\text{-Alg}$*, then* $\mathrm{Eqs}(\mathscr{C})$ *has coequalisers.*

Proof sketch of Lemma 3.9. (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

$$L \text{ and } R : (\textstyle\coprod_{i \in I} \Sigma_i)\text{-Alg} \to (\coprod_{i \in I} \Gamma_i)\text{-Alg}$$

map $\alpha : \coprod_{i \in I} \Sigma_i A \to A$ to $[L_i(\alpha \cdot \iota_i)]_{i \in I}$ and $[R_i(\alpha \cdot \iota_i)]_{i \in I} : \coprod_{i \in I} \Gamma_i A \to A$ respectively.

(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 of the following two functorial terms

$$\Sigma\text{-Alg} \longrightarrow \hat{\Sigma}\text{-Alg} \underset{T_2}{\overset{T_1}{\rightrightarrows}} \hat{\Gamma}\text{-Alg} \hookrightarrow \Gamma\text{-Alg}$$

The coequaliser $\hat{\Sigma} \to \hat{\Sigma}'$ is the inclusion functor: $\hat{\Sigma}'\text{-Alg} \hookrightarrow \hat{\Sigma}\text{-Alg}$. □

Theorem 3.10. *The full subcategory* $\mathrm{Eqs}_\omega(\mathscr{C}) \subseteq \mathrm{Eqs}(\mathscr{C})$ *of equational systems whose functorial signature and context preserve colimits of* ω*-chains is cocomplete if* $\mathscr{C}$ *is cocomplete.*

Proof of Theorem 3.10. For such equational systems $\hat{\Sigma}$, there are reflections $\Sigma\text{-Alg} \underset{\perp}{\rightleftarrows} \hat{\Sigma}\text{-Alg}$ by Theorem 3.2. Then cocompleteness follows from the lemma above since colimits can be constructed using coequalisers and coproducts. □

Lemma 3.11. *Let* $\hat{\Sigma} \in \mathrm{Eqs}(\mathscr{C})$ *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 following is a pushout diagram of* T_1 *and* T_2*:*

$$\begin{array}{ccc} \hat{\Sigma} & \xrightarrow{\quad T_2 \quad} & \hat{\Sigma} \curlywedge_{\mathrm{op}} \Phi_2 \curlywedge_{\mathrm{eq}} E_2 \\ {\scriptstyle T_1}\downarrow & & \downarrow \\ \hat{\Sigma} \curlywedge_{\mathrm{op}} \Phi_1 \curlywedge_{\mathrm{eq}} E_1 & \longrightarrow & \hat{\Sigma} \curlywedge_{\mathrm{op}} (\Phi_1 + \Phi_2) \curlywedge_{\mathrm{eq}} E' \end{array}$$

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* $\alpha_i : (\Sigma + (\Phi_1 + \Phi_2))\text{-Alg} \to (\Sigma + \Phi_i)\text{-Alg}$ *is the projection functor.*

Proof of Lemma 3.11. First of all, there are evident inclusion translations P_i (which are the unlabelled arrows in the pushout diagram):

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

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

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

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

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

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

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

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

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

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

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

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 $(M \square -)$-Mon extended with equations

$$\cdot \vdash \mathrm{op}(\eta^M, \eta^\tau) = \eta^\tau : \tau$$

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

saying that $\mathrm{op} : M \square \tau \to \tau$ 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)

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

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

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 $(\hat{M}\text{-Act})$-Alg is just the coslice category $\hat{M}/\mathrm{Mon}(\mathscr{E})$. Thus the initial algebra of $\hat{M}$-Act is $\hat{M}$ as required.
- Starting from a theory $\langle \hat{\Sigma}, T_\Sigma \rangle \in \mathrm{ALG}(\mathscr{E})$ where

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

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

$$S \square A \xrightarrow{S \square \eta^{\hat{\Sigma}^*}} S \square \hat{\Sigma}^* \square A \xrightarrow{\alpha^{\hat{\Sigma}^*}} \hat{\Sigma}^* A \xrightarrow{\alpha} A$$

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

$$\hat{\Sigma}^* \square A \xrightarrow{(\!|\beta|\!) \square A} A \square A \xrightarrow{\mu^A} A$$

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

□

Lemma B.1. *Let $\mathscr{E}$ be a cocordial monoidal category, and $\tilde{\Gamma} \in \mathrm{ALG}(\mathscr{E})$ and $\tilde{\Sigma} \in \mathrm{Mon/Eqs}_\omega(\mathscr{E})$. Then $\mathrm{Mon/Eqs}_\omega(\tilde{\Gamma}, \tilde{\Sigma})$ is in natural bijection to monoid morphisms $\tilde{\Gamma}^* \to \tilde{\Sigma}^*$.*

Proof of Lemma B.1. For one direction of the bijection ϕ, given a translation $T : \tilde{\Gamma} \to \tilde{\Sigma}$, T sends the initial algebra $\tilde{\Sigma}^*$ of $\tilde{\Sigma}$ to a $\tilde{\Gamma}$-algebra carried by $\tilde{\Sigma}^*$. Then by the initiality of $\tilde{\Gamma}^*$, there is a $\tilde{\Gamma}$-homomorphism, which is also a monoid morphism, $u : \tilde{\Gamma}^* \to \tilde{\Sigma}^*$. We set the $\phi(T) = u$.

For the back direction of the bijection ϕ, given a monoid morphism $h : \tilde{\Gamma}^* \to \tilde{\Sigma}^*$. We define a translation $T : \tilde{\Gamma} \to \tilde{\Sigma}$, i.e. a functor $T : \tilde{\Sigma}\text{-Alg} \to \tilde{\Gamma}\text{-Alg}$. Recall that $\tilde{\Gamma} \in \mathrm{ALG}(\mathscr{E})$ must be of the form $\hat{\Gamma} = (G \square -)\text{-Mon} \upharpoonleft_{\mathrm{eq}} (\mathrm{K}_B \vdash L = R)$, 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

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

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

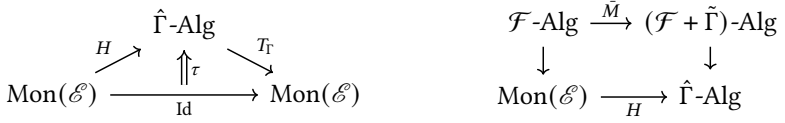
It can be checked that ϕ is a bijection. □

Theorem 4.4. *Let $\mathscr{E}$ be a cocordial monoidal category. (i) The category* $\mathrm{ALG}(\mathscr{E})$ *is a coreflective subcategory of* $\mathrm{Mon/Eqs}_\omega(\mathscr{E})$*, i.e. there is an adjunction* $\mathrm{ALG}(\mathscr{E}) \underset{\tau}{\leftrightarrows} \mathrm{Mon/Eqs}_\omega(\mathscr{E})$*. (ii) Moreover, the coreflector $\lfloor - \rfloor$ preserves initial algebras: for every $\langle \hat{\Sigma} \in \mathrm{Eqs}_\omega(\mathscr{E}), T : \mathrm{Mon} \to \hat{\Sigma} \rangle$, 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 $\langle \hat{\Sigma}, T_\Sigma \rangle \in \mathrm{Mon/Eqs}_\omega(\mathscr{E})$ always has an initial algebra carried by $\hat{\Sigma}^* \in \mathscr{E}$ by Theorem 3.10, and $\hat{\Sigma}^*$ carries a monoid structure by the translation $T_\Sigma : \mathrm{Mon} \to \hat{\Sigma}$. The coreflector $\lfloor - \rfloor$ maps the theory $\langle \hat{\Sigma}, T_\Sigma \rangle$ to $\hat{\Sigma}^*$-Act $\in \mathrm{ALG}(\mathscr{E})$ as in the proof of Theorem 4.3. For every theory $\langle \hat{\Gamma}, T_\Gamma \rangle \in \mathrm{ALG}(\mathscr{E})$, by Lemma B.1, each translation in the hom-set $\mathrm{Mon/Eqs}_\omega(\langle \hat{\Gamma}, T_\Gamma \rangle, \langle \hat{\Sigma}, T_\Sigma \rangle)$ is equivalently a monoid morphism $\hat{\Gamma}^* \to \hat{\Sigma}^*$, which is also equivalently a translation in $\mathrm{ALG}(\langle \hat{\Gamma}, T_\Gamma \rangle, \lfloor \langle \hat{\Sigma}, T_\Sigma \rangle \rfloor)$ by Theorem 4.3.

For part (ii) of the theorem, the coreflector maps each $\tilde{\Sigma} \in \mathrm{Mon/Eqs}_\omega(\mathscr{E})$ to the theory $\tilde{\Sigma}^*$-Act. It is not difficult to show that the category algebras of $\tilde{\Sigma}^*$-Act is equivalent to the coslice category $\tilde{\Sigma}^*/\mathrm{Mon}(\mathscr{E})$ of monoids under $\tilde{\Sigma}^*$. So the initial algebra of $\tilde{\Sigma}^*$-Act is still the same monoid $\tilde{\Sigma}^*$. □

Theorem 6.2. *For each $\tilde{\Gamma} = \langle \hat{\Gamma}, T_\Gamma \rangle \in \mathrm{ALG}(\mathscr{E})$ over a cocordial monoidal category $\mathscr{E}$, a functor $H : \mathrm{Mon}(\mathscr{E}) \to \hat{\Gamma}$-Alg and a $\tau : \mathrm{Id} \to T_\Gamma \circ H$ can be extended to a modular model $\bar{M}$ of $\tilde{\Gamma}$ such that the diagram on the right below commutes, and $\bar{M}$ has a lifting $\bar{I}_{\langle \langle \hat{\Sigma}, T_\Sigma \rangle,\, A,\, \alpha \rangle} = \tau_{\langle A,\, T_\Sigma \alpha \rangle}$:*



(22)

where the unlabelled vertical arrows are the evident projection functors.

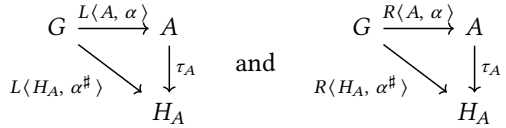
Proof sketch of Theorem 6.2. For each object $\langle \hat{\Sigma}, T_\Sigma, A, \alpha \rangle$ of $\mathcal{F}$-Mon with

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

we define M to send it to a $(\mathcal{F} + \hat{\Gamma})$-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

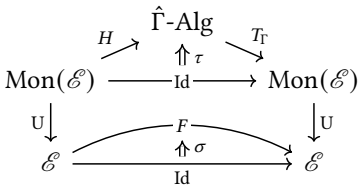
$$\alpha^\sharp = [\![s : S, h : H_A \vdash \mu^H(\tau_A(\alpha_S(s, \eta^A)),\ h) : H_A]\!] \tag{24}$$

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



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 6.5. *For each $\langle \hat{\Gamma}, T_\Gamma\rangle \in \mathrm{SCP}(\mathscr{E})$ over a cocordial and* closed *monoidal category $\mathscr{E}$, a functor $H : \mathrm{Mon}(\mathscr{E}) \to \hat{\Gamma}\text{-Alg}$ 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 $\mathrm{U} \circ T_\Gamma \circ H = F \circ \mathrm{U}$ and $\tau \circ \mathrm{U} = \sigma \circ U$ can be extended to a modular model M of $\langle \hat{\Gamma}, T_\Gamma\rangle$ with a lifting l such that (22) commutes and $l_{\langle\langle \hat{\Sigma}, T_\Sigma\rangle, A, \alpha\rangle} = \tau_{\langle A, T_\Sigma\alpha\rangle}$.*



Proof sketch of Theorem 6.5. Compared to Theorem 6.2, 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

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

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):

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

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 6.2. □

Theorem 6.10 (Dependent Combination)**.** *Let $\mathscr{E}$ be a $\langle \mathrm{Endo}_f\mathbf{Set}, \star, \mathrm{Id}\rangle$ and $\mathcal{F}$ be $\mathrm{ALG}(\mathscr{E})$ or $\mathrm{SCP}(\mathscr{E})$. For each $\tilde{\Gamma} \in \mathcal{F}$, every ordinary model $\hat{A} \in \tilde{\Gamma}\text{-Alg}$ induces a 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 6.10. 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 $s : S \star (A \circ B) \to (S \star A) \circ B$:

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

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 $t : G \star (A \circ B) \to A \circ (G \star B)$:

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

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