Modular Models of Monoids with Operations

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

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

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

1 INTRODUCTION

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

Additionally, variations of monads are proposed to model more forms of computational effects, including *arrows* [Hughes 2000; Jacobs et al. 2009], *applicative functors* [Mcbride and Paterson 2008; Paterson 2012], *parameterised monads* [Atkey 2009], *graded monads* [Katsumata 2014]. All these notions can be unified in the framework of *monoids in monoidal categories*, leading up to the principle of *notions of computation as monoids* [Pieters et al. 2020; Rivas and Jaskelioff 2017].

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

Syntactic modularity is where (the syntactic specification of) two languages Σ_1 and Σ_2 can be combined into a larger language $\Sigma_1 +_{\text{syn}} \Sigma_2$.

Semantic modularity is where semantic models M_1 and M_2 for the sub-languages can be combined into a model $M_1 +_{\text{sem}} M_2$ of the larger language $\Sigma_1 +_{\text{syn}} \Sigma_2$.

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

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

Effect Handlers. Notably, *effect handlers*, introduced by Plotkin and Pretnar [2013] and further developed by many people, achieve both kinds of modularity, which explains their quick adoption.

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.

In this approach, effectful operations are described by an algebraic theory (sometimes without any equations), whose free-algebra monad is used as the monad for effectful computations. Effect handlers are (not necessarily free) algebras of this theory, so they can 'handle' effectful computations, i.e. the free algebra, using the unique homomorphism out of the free algebra. Algebraic theories and handlers are both composable so syntactic and semantic modularity are both achieved.

However, effectful operations in this framework *necessarily* fall into a kind of effectful operation known as *algebraic operations* [Plotkin and Power 2003]. An algebraic operation on a monad $M : \mathscr{C} \to \mathscr{C}$ is a natural transformation $\alpha : A \circ M \to M$ for an endofunctor $A : \mathscr{C} \to \mathscr{C}$, typically a polynomial functor, such that it is compatible with monad multiplication:

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

which intuitively says that the operation α commutes with sequential composition of computations. Not all effectful operations have this property though: for example, exception catching in Haskell $catch :: IO\ a \to (Exc \to IO\ a) \to IO\ a$ does not satisfy $(catch\ p\ h) \gg\!= k = catch\ (p \gg\!= k)\ (h \gg k)$ since the left-hand side only catches the exception in p while the right-hand side catches the exception in p and k. These non-algebraic operations *can* be programmed as handlers [Plotkin and Pretnar 2013], but they do not have the benefit of (syntactic or semantic) modularity, since as handlers they themselves cannot be modularly handled [Wu et al. 2014; Yang et al. 2022].

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

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

where the isomorphism is by the co-Yoneda lemma. Typically, the functor A is $(-)^n$ for some natural number n. The intuition is that s is an operation delimiting n scopes: the coend $\int^X$ is like an existential type $\exists X$, and $A(MX) = (MX)^n$ is the computation inside the scopes, returning some type X, and $(M-)^X$ is the computation after these scopes. For example, the operation of *exception catching* delimits two scopes, one for 'try' and the other for 'catch', so $A = (-)^2 \cong (- \times -)$.

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

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

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

Overview. Motivated by the lack of semantic modularity in existing frameworks, the present paper has two aims/parts: the first one is to develop a unifying account of equational theories of (algebraic and non-algebraic) effectful operations; the second one is to develop a framework of *modular models* that provide semantic modularity for non-algebraic operations. These two parts together achieve both modularities in the style of effect handlers, but for a wider range of operations.

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

The main definition of the second part (§5–6) is *modular models* M of some theory $\ddot{\Psi}$ in a theory family $\mathcal{F}$. Every M is basically a family of functors $\ddot{\Sigma}$-**Alg** $\to$ $(\ddot{\Sigma} + \ddot{\Psi})$-**Alg** (oplax-) natural in $\ddot{\Sigma} \in \mathcal{F}$. In plain words, a modular model sends monoids with operations to monoids with *more* operations, so semantic modularity is *built into* the definition of modular models. Then the question becomes how to construct modular models, for which we show several constructions and examples in §6.

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

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

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

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

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

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

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

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

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

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

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

introduce *modular models* of a theory and show how they can be used for interpreting syntax. In §6, we show general constructions and examples of modular models.

Along these lines, we make the following contributions:

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

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

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

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

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

2 MONOIDS, MONOIDAL CATEGORIES, AND A METALANGUAGE

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

2.1 Notions of Computation as Monoids in Monoidal Categories

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

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

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

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

$$\begin{array}{ccccc} (M \square M) \square M & \xrightarrow{\alpha_{M,M,M}} & M \square (M \square M) & & \\ \downarrow{\scriptstyle \mu \square M} & & \downarrow{\scriptstyle M \square \mu} & & \\ M \square M & \xrightarrow[\mu]{} M & \xleftarrow[\mu]{} & M \square M & \end{array} \qquad \begin{array}{ccccc} I \square M & \xleftarrow{\lambda_M^{-1}} & M & \xrightarrow{\rho_M^{-1}} & M \square I \\ \downarrow{\scriptstyle \eta \square M} & & \| & & \downarrow{\scriptstyle M \square \eta} \\ M \square M & \xrightarrow[\mu]{} M & \xleftarrow[\mu]{} & & M \square M \end{array} \tag{3}$$

Below we review several examples of monoidal categories, in which monoids model different flavours of *notions of computations*, and will serve as the main application of the theory developed in this paper. Should any of the examples be unfamiliar to the reader, they can be glossed over.

Monads. The category $\mathbf{Endo}(\mathscr{C})$ of endofunctors $\mathscr{C} \to \mathscr{C}$ on a category $\mathscr{C}$ can be turned into a monoidal category by equipping it with functor composition $F \circ G$ as the monoidal product and

the identity functor $\mathrm{Id} : \mathscr{C} \to \mathscr{C}$ as the unit. Monoids in this category are called *monads* on $\mathscr{C}$, and they are used to model *computational effects*, also called *notions of computation*, in programming languages [Moggi 1989b, 1991], where the unit $\eta : \mathrm{Id} \to M$ is understood as embedding *pure values* into computations, and the multiplication $\mu : M \circ M \to M$ is understood as flattening *computations of computations* into computations by sequentially executing them. The understanding of μ as sequential composition is better exhibited by the following co-Yoneda isomorphism:

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

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

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

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

$$\mathbf{Endo}_f(\mathbf{Set}) \underset{-\circ V}{\overset{\mathbf{Lan}_V -}{\underset{\longrightarrow}{\overset{\longleftarrow}{\cong}}}} \mathbf{Set}^{\mathbf{Fin}} \tag{4}$$

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

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

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

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

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

Sometimes the semantics of a programming language must be given in a category other than $\mathbf{Set}$. Thus it is useful to generalise $\mathbf{Endo}_f(\mathbf{Set}) \cong \mathbf{Set}^{\mathbf{Fin}}$: we can replace $\mathbf{Set}$ with any *locally κ-presentable* (lκp) category $\mathscr{C}$ for a regular cardinal κ, $\mathbf{Fin}$ with the subcategory $\mathscr{C}_\kappa$ of *κ-presentable objects* in $\mathscr{C}$, and finitary functors with *κ-accessible* functors $\mathbf{Endo}_\kappa(\mathscr{C})$ [Adámek and Rosicky 1994]. This results in a cocomplete closed monoidal category $\langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$, on which $\circ$ is κ-accessible.

Examples of lκp categories include presheaf categories $\mathbf{Set}^{\mathscr{D}}$ for small categories $\mathscr{D}$ and $\kappa = \aleph_0$, the category $\mathbf{Cat}$ of small categories for $\kappa = \aleph_0$, and the category $\omega\mathbf{Cpo}$ of ω-complete partial orders for $\kappa = \aleph_1$. The category of $\omega\mathbf{Cpo}$ is particularly relevant to us since it can be used for modelling languages with recursion. Moreover, when $\mathscr{C}$ is lκp, it is automatically lλp for any $\lambda > \kappa$.

Locally κ-presentable categories provide a nice setting for doing algebraic theories and are general enough for various applications in programming languages. In this paper we use lκp categories as a condition to ensure the existence of free monoids with operations, but readers unfamiliar with them may think with the special case $\mathscr{C} = \mathbf{Set}$ whenever lκp categories are mentioned.

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 $\mathbf{Set}$ of sets is a closed monoidal category in this way. Monoids in $\mathbf{Set}$ are precisely the usual notion of monoids in algebra, such as the set of lists with concatenation and empty list.

The category $\mathbf{Endo}_\kappa(\mathscr{C})$ is also cartesian closed whenever $\mathscr{C}$ is lκp and cartesian closed. The cartesian unit and product in $\mathbf{Endo}_\kappa(\mathscr{C})$ are defined pointwise: $1n = 1_{\mathscr{C}}$ and $(F \times G)n = Fn \times Gn$. A computational interpretation of cartesian monoids in $\mathbf{Endo}_\kappa(\mathscr{C})$ is that they model *notions of independent computations*: the cartesian product $M \times M \to M$ composes two computations that have no dependency and return the same type of values, whereas monad multiplication $M \circ M \to M$ composes a computation with another that depends on the result of the former.

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

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

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

Monoids for $\langle \star, \mathrm{Id} \rangle$ are called *applicative functors* or simply *applicatives* [Mcbride and Paterson 2008; Paterson 2012]. A practical application of them is in build systems [Mokhov et al. 2018], since usually the result of a building task does not affect what the next building task is.

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

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

However, to give semantics to programming languages with *non-linear* variable contexts, what we need is slightly stronger: for all objects $\Gamma \in \mathscr{C}$ (thought of as variable contexts) and pairs of arrows $f : \Gamma \times A \to MB$ and $g : \Gamma \times B \to MC$ (two computations under the context Γ), we would like to have an arrow $\Gamma \times A \to MC$. The structure of monads $\langle M, \mu, \eta \rangle$ is not sufficient for doing this; we need additionally a natural transformation $s_{\Gamma,B} : \Gamma \times MB \to M(\Gamma \times B)$, giving rise to the composite

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

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

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

$$\begin{array}{ccc} 1 \times FB & & \\ {\scriptstyle s_{1,B}}\downarrow & \searrow^{\lambda_{FB}} & \\ F(1 \times B) & \xrightarrow[F\lambda_B]{} & FB \end{array} \qquad \begin{array}{ccccc} (\Gamma' \times \Gamma) \times FB & \xrightarrow{\alpha_{\Gamma',\Gamma,FB}} & \Gamma' \times (\Gamma \times FB) & \xrightarrow{\Gamma' \times s_{\Gamma,B}} & \Gamma' \times F(\Gamma \times B) \\ {\scriptstyle s_{\Gamma' \times \Gamma, B}}\downarrow & & & & \downarrow{\scriptstyle s_{\Gamma',\Gamma \times B}} \\ F((\Gamma' \times \Gamma) \times B) & & \xrightarrow[F\alpha_{\Gamma',\Gamma,B}]{} & & F(\Gamma' \times (\Gamma \times B)) \end{array} \tag{7}$$

where λ and α are the left unitor and associator for the cartesian monoidal structure $\langle \times, 1 \rangle$ on $\mathscr{C}$. Moreover, *strong natural transformations* between strong functors $\langle F, s^F \rangle$ and $\langle G, s^G \rangle$ are natural transformations $\tau : F \to G$ such that $s^G_{\Gamma,B} \cdot (\Gamma \times \tau_B) = \tau_{\Gamma \times B} \cdot s^F_{\Gamma,B} : \Gamma \times FB \to G(\Gamma \times B)$. Strong endofunctors on $\mathscr{C}$ and strong natural transformations can be collected into a category $\mathbf{Endo}_s(\mathscr{C})$.

The category $\mathbf{Endo}_s(\mathscr{C})$ has a monoidal structure $\langle \circ_s, \mathrm{Id}_s \rangle$: Id_s is the identity functor equipped with the identity strength, and $\circ_s$ is the composition of strong functors:

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

Strong monads on $\mathscr{C}$ are precisely monoids in the monoidal category $\langle \mathbf{Endo}_s(\mathscr{C}), \circ_s, \mathrm{Id}_s \rangle$. When $\mathscr{C}$ is cartesian *closed*, strong functors/natural transformation are the same thing as $\mathscr{C}$*-enriched* functors/natural transformations [Kock 1972; McDermott and Uustalu 2022b].

Similar to the setting of κ-accessible monads, it is beneficial to restrict the category $\mathbf{Endo}_s(\mathscr{C})$ when considering algebraic theories over it: we denote by $\mathbf{Endo}_{s\kappa}(\mathscr{C})$ the full subcategory of $\mathbf{Endo}_s(\mathscr{C})$ that contains strong functors whose underlying functors are κ-accessible. When $\mathscr{C}$ is *lκp as a cartesian closed category*, which means that $\mathscr{C}$ is lκp and cartesian closed, and that the κ-presentable objects of $\mathscr{C}$ are closed under finite products, the category $\mathbf{Endo}_{s\kappa}(\mathscr{C})$ is also lκp and has a closed monoidal structure of functor composition and the identity functor. The primary non-trivial example of such setting is $\mathscr{C} = \omega\mathbf{Cpo}$ for modelling programming languages with general recursion. We refer the reader to Kelly and Power [1993, §4] and Kelly [1982] for details.

Graded Monads. A generalisation of monads is to index the monad with some *grades* that track quantitative information about the effects performed by a computation [Katsumata 2014; Katsumata et al. 2022; McDermott and Uustalu 2022a]. For example, the grades can be a set of operations that a computation may invoke, or it can be the number of nondeterministic choices that a computation makes. Precisely, let $\langle \mathbb{G}, \cdot, 1 \rangle$ be any small strict monoidal category, whose objects are called grades. A *finitary* $\mathbb{G}$*-graded monad* [Kura 2020] is a functor $M : \mathbb{G} \to \mathbf{Endo}_f(\mathbf{Set})$ equipped with

$$\eta : \mathrm{Id} \to M1 \qquad \mu_{a,b} : (Ma) \circ (Mb) \to M(a \cdot b) \quad \text{for all } a, b \in \mathbb{G} \tag{8}$$

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

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

$$I = \coprod_{\mathbb{G}(1,-)} \mathrm{Id} \qquad F \star G = \int^{a,b \in \mathbb{G}} \coprod_{\mathbb{G}(a \cdot b,-)} (Fa \circ Gb). \tag{9}$$

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

2.2 A Metalanguage for Monoidal Categories

When the monoidal category gets complex, commutative diagrams like those in (3) can be unwieldy, we will use a typed calculus introduced by Jaskelioff and Moggi [2010] as a *metalanguage* to

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

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

Fig. 1. Well typed terms for the metalanguage.

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

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

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

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

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

Fig. 3. Equational theory of the metalanguage.

denote constructions in monoidal categories, in the same way that λ-calculus can be used to denote constructions in cartesian closed categories. The use of the calculus not only provides a convenient syntax, but also provides useful intuition as if we were working in the category of sets.

Types and Terms. A *metalanguage for monoidal categories* $\mathcal{L} = \langle \mathsf{Ba}, \mathsf{Pr}, \mathsf{Ax} \rangle$ is specified by three components. The set Ba is a set of base types, ranged over by α. The types of $\mathcal{L}$ are inductively generated by the rule $A, B ::= \alpha \mid A \square B \mid \mathsf{I}$. The set Pr is a set of primitive operations, ranged over by f, each associated with two types $f : A \to B$. The well typed terms of $\mathcal{L}$ are generated by the typing rules in Figure 1 (there is no need to consider raw terms for our purposes). The type system is a substructural one, 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 the cut rule below is admissible, in which $t[u/x]$ is substituting u for x in t:

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

Lastly, Ax is a set of axioms, each $\langle \Gamma \vdash t_l : A,\ \Gamma \vdash t_r : A \rangle$ is a pair of terms under the same context.

Interpretation. An interpretation $[\![-]\!]$ of the metalanguage $\mathcal{L} = \langle \mathsf{Ba}, \mathsf{Pr}, \mathsf{Ax} \rangle$ in a monoidal category $\mathscr{C}$ consists of (i) an assignment of $\mathscr{C}$-objects $[\![\alpha]\!] \in \mathbf{Ob}(\mathscr{C})$ to each base type $\alpha \in \mathsf{Ba}$, which determines the denotation of all types and contexts $\mathscr{C}$ as follows:

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

and (ii) an assignment of $\mathscr{E}$-arrows $[\![f]\!] : [\![A]\!] \to [\![B]\!]$ to each primitive operation $f : A \to B \in \mathrm{Pr}$, which determines the denotation of all terms as in Figure 2. The interpretation must make $[\![t_l]\!] = [\![t_r]\!]$ for all axioms $\langle t_l, t_r \rangle \in \mathrm{Ax}$. Moreover, we write $[\![-]\!]_{\{x \mapsto X\}}$ for the interpretation that maps the basic type or primitive operation x to X and everything else the same as $[\![-]\!]$.

Equational Theory. The equational theory $\Gamma \vdash t_1 \equiv t_2 : A$ of the metalanguage $\mathcal{L}$ is generated by the rules in Figure 3, the axioms in Ax, and the usual rules for equivalence and congruence under all term formers. The rules in Figure 3 characterise the syntactic multicategory of the metalanguage (quotiented by the equational theory) as a *representable multicategory*, thus corresponding to a monoidal category [Hermida 2000]. Analogous to the classical correspondence between simply typed λ-calculus and cartesian closed categories [Lambek and Scott 1986], the equational theory is sound for all interpretations: $\Gamma \vdash t_1 \equiv t_2 : A$ implies $[\![t_1]\!] = [\![t_2]\!]$. The converse is also true, following from the fact that the syntactic multicategory corresponds to the free monoidal category over $\mathcal{L}$, but we do not need (and do not prove) this result in this paper.

Internal Language. Given a monoidal category $\mathscr{E}$, its *internal language* $\mathcal{L}(\mathscr{E})$ has all objects $\mathscr{E}$ as basic types and all arrows as primitive operations. These basic types and primitive operations have a canonical interpretation in $\mathscr{E}$—everything is interpreted as itself. The axioms of $\mathcal{L}(\mathscr{E})$ are *all* pairs of terms that have the same denotation. Therefore the language $\mathcal{L}(\mathscr{E})$ provides a sound and complete way to reason about $\mathscr{E}$: two terms in $\mathcal{L}(\mathscr{E})$ satisfy $t_1 \equiv t_2$ if and only if they are equal under the canonical interpretation in $\mathscr{E}$. In the rest of this paper, whenever we work with a monoidal category $\mathscr{E}$, we often use $\mathcal{L}(\mathscr{E})$ to denote and reason about constructions in $\mathscr{E}$.

Example 2.1. In the metalanguage with a base type M and a primitive operation $\mu : M \boxtimes M \to M$. The first diagram in the laws of monoids (3) can be represented by the following axiom (throughout this paper, we write $\Gamma \vdash t_1 = t_2 : A$ for the equational axiom consisting of t_1 and t_2):

$$x : M, y : M, z : M \vdash \mu(\mu(x, y), z) \;=\; \mu(x, \mu(y, z)) : M \tag{10}$$

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

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

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

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

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

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

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

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

as well as their β and η rules. 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 monoidal closed category. If some type M together with two terms $(x : M, y : M \vdash \mu : M)$ and $(\cdot \vdash \eta : M)$ in $\mathcal{L}(\mathscr{E})$ 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

$$f : M/M,\ g : M/M \vdash \lambda x.\, f\ (g\ x) : M/M. \tag{11}$$

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

This elementary result has surprisingly many applications in functional programming for optimisation, because the multiplication (11) is usually a kind of function composition with $O(1)$ time complexity, regardless of the possibly expensive multiplication μ. When M is a free monoid in $\langle \mathbf{Set}, \times, 1 \rangle$, i.e. a list, this optimisation is known as *difference lists* [Hughes 1986]. When $\mathscr{E} = \langle \mathbf{End}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$, this optimisation is known as *codensity transformation* [Hinze 2012]. We will also use this fact later for constructing modular models in Theorem 6.4.

3 EQUATIONAL SYSTEMS AND TRANSLATIONS

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

Therefore we need a way to talk about monoids equipped with additional operations as *equational theories* and their models, especially their *free models*, which are practically important since they play the role of abstract syntax. Moreover, we would like to be able to combine such theories.

To fulfil these needs, we use Fiore and Hur [2007, 2009]'s *equational systems* to formulate equational theories (§3.1) and their theorem for constructing free models (§3.2), which we first recap below. We also show how the metalanguage from §2 can be used to present equational systems syntactically. Then we recap the equational system of Σ-*monoids*, which is the theory of monoids with operations (§3.3). Lastly, we introduce *functorial translations* between equational systems, making them a category, and discuss the existence of colimits in this category (§3.4).

3.1 Equational Systems

An equational theory consists of the *signature* and *equations* of its operations. A concise way to specify a signature on a category $\mathscr{C}$ is just 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 signature functor of the *theory of monoids* in a monoidal category $\mathscr{E}$ is $\Sigma_{\mathbf{Mon}} = (- \square -) + I$. A $\Sigma_{\mathbf{Mon}}$-algebra $\langle A, \alpha \rangle$ is an object A with an arrow $\alpha : (A \square A) + A \to A$, or equivalently two arrows $A \square A \to A$ and $I \to A$.

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

Equations on a signature $\Sigma : \mathscr{C} \to \mathscr{C}$ are usually presented as commutative diagrams (like the first diagram in (3) for associativity), which are pairs of paths from ΓA to A built out a formal arrow $\Sigma A \to A$ and ΓA is the starting node of the diagram, which may be called the *context* of the diagram. One way to formulate such a diagram is as a pair of functors $L, R : \Sigma\text{-}\mathbf{Alg} \to \Gamma\text{-}\mathbf{Alg}$. For example, the down-right path of the associativity diagram (3) can be represented by a functor $(- \square -)\text{-}\mathbf{Alg} \to ((- \square -) \square -)\text{-}\mathbf{Alg}$ that sends $\langle A, \mu : A \square A \to A \rangle$ to $\langle A, \mu \cdot (\mu \square M) \rangle$. Such a functor $L : \Sigma\text{-}\mathbf{Alg} \to \Gamma\text{-}\mathbf{Alg}$ encoding half of a commutative diagrams always satisfies $\mathbf{U}_\Gamma \circ L = \mathbf{U}_\Sigma$.

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

An *algebra* or a *model* of $\dot{\Sigma}$ 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 $\dot{\Sigma}$-algebras is denoted by $\dot{\Sigma}$-**Alg**.

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

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

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

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

(ii) To give a *functorial term* $\Sigma\text{-}\mathbf{Alg} \to \Gamma\text{-}\mathbf{Alg}$ for functors $\Sigma, \Gamma : \mathscr{E} \to \mathscr{E}$ given by type expressions S_τ and G_τ in the way above, it is sufficient to write a term $x : G_\tau \vdash t : \tau$ in the language $\mathcal{L}(\mathscr{E})$ extended with a base type τ and a primitive operation $\mathrm{op} : S_\tau \to \tau$. Then the term $x : G_\tau \vdash t : \tau$ induces a functor $T : \Sigma\text{-}\mathbf{Alg} \to \Gamma\text{-}\mathbf{Alg}$ such that

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

By structural induction on typing derivations in the style of Reynolds [1983]'s abstraction theorem, it is possible to show that $\mathbf{U}_\Gamma \circ T = \mathbf{U}_\Sigma$ as required in Definition 3.1. For example, the two terms (10) denote a pair of functorial terms $(- \,\square\, -)\text{-}\mathbf{Alg} \to (- \,\square\, - \,\square\, -)\text{-}\mathbf{Alg}$.

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

$$\mathbf{Mon} = (\Sigma_{\mathbf{Mon}} \triangleright \Gamma_{\mathbf{Mon}} \vdash L_{\mathbf{Mon}} = R_{\mathbf{Mon}}) \tag{12}$$

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

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

and the functorial equation $L_{\mathbf{Mon}} = R_{\mathbf{Mon}}$ is given by two terms in $\mathcal{L}(\mathscr{E})$ with a new primitive operation $\mathrm{op} : \tau \,\square\, \tau + \mathsf{I} \to \tau$ as follows: first we define two terms for using op more convenient:

$$\mu = (x : \tau, y : \tau \vdash \mathrm{op}\ (\mathsf{inj}_1(x, y)) : \tau) \qquad \eta = (\cdot \vdash \mathrm{op}\ (\mathsf{inj}_2 *) : \tau)$$

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

Then we define the following terms in which $\mu[t_1, t_2]$ is shorthand for $\mu[t_1/x, t_2/y]$:

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

Finally $L_{\mathbf{Mon}} = R_{\mathbf{Mon}}$ is given by $x : (\tau \,\square\, \tau \,\square\, \tau) + \tau + \tau \vdash [l_1, l_2, l_3] = [r_1, r_2, r_3] : \tau$. A model of the equational system **Mon** is exactly a monoid in $\langle \mathscr{E}, \square, I \rangle$.

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

$$\dot{\Sigma} \curvearrowleft_{\mathrm{op}} \Sigma' \coloneqq (\Sigma + \Sigma' \ \triangleright\ \Gamma \ \vdash\ L \circ \pi = R \circ \pi)$$

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

$$\dot{\Sigma} \curvearrowleft_{\mathrm{eq}} (\Gamma' \vdash L' = R') \coloneqq (\Sigma \ \triangleright\ \Gamma + \Gamma' \ \vdash\ [L, L'] = [R, R']).$$

3.2 Free Algebras of Equational Systems

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

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

Given any model $\langle A, \alpha \rangle$ of $\dot{\Sigma}$ and $g : X \to A$, the morphism $\phi(g) : \mathbf{Free}\ X \to \langle A, \alpha \rangle$ 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 3.3 (Fiore and Hur [2009])**.** *For all equational systems $\dot{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ 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*

$$\dot{\Sigma}\text{-}\mathbf{Alg} \underset{\longrightarrow}{\overset{\longleftarrow}{\hookrightarrow}}{\scriptstyle \bot} \Sigma\text{-}\mathbf{Alg} \underset{\longrightarrow}{\overset{\longleftarrow}{\longrightarrow}}{\scriptstyle \bot} \mathscr{C} \tag{13}$$

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

Notation 3.4. We denote the composite adjunction of (13) by $\mathbf{F}_{\dot{\Sigma}} \dashv \mathbf{U}_{\dot{\Sigma}} : \dot{\Sigma}\text{-}\mathbf{Alg} \to \mathscr{C}$, or simply $\mathbf{F} \dashv \mathbf{U}$ when $\dot{\Sigma}$ is understood. Moreover, the initial $\dot{\Sigma}$-algebra is denoted by $\langle \mu\dot{\Sigma}, \alpha^{\dot{\Sigma}} : \Sigma\mu\dot{\Sigma} \to \mu\dot{\Sigma} \rangle$.

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 $\dot{\Sigma}$-algebra.

Example 3.5. When $\mathscr{E}$ is cocomplete and $\square : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ preserves α-chains for some limit ordinal α, then Theorem 3.3 is applicable to the equational system **Mon** (12). Moreover, when $\mathscr{E}$ is closed, there is a simple formula for free monoids: for every $A \in \mathscr{E}$, the free monoid over A is the initial algebra $\mu X.I + A \square X$ equipped with appropriate monoid operations [Fiore 2008]. This formula is useful in practice: when $\mathscr{E}$ is $\langle \mathbf{Set}, \times, 1 \rangle$, it is exactly the usual definition $\mu X.1 + A \times X$ of A-lists; and when $\mathscr{E}$ is $\langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ or $\langle \mathbf{Endo}_\kappa(\mathbf{Set}), \star, \mathrm{Id} \rangle$ in §2.1, this gives formulas for free monads and free applicatives that are suitable for implementation [Rivas and Jaskelioff 2017].

3.3 Equational Systems for Monoids with Operations

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

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

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 (7). To denote Σ and θ syntactically, we extend the metalanguage with a type constructor Σ and the following typing rule

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

Then the equational system Σ-**Mon** of Σ-*monoids* extends the theory **Mon** of monoids (12) with a new operation $\mathrm{op} : \Sigma\tau \to \tau$ and a new equation $L_{\Sigma\text{-}\mathbf{Mon}} = R_{\Sigma\text{-}\mathbf{Mon}}$:

$$\Sigma\text{-}\mathbf{Mon} = (\mathbf{Mon} \upharpoonleft_{\mathrm{op}} \Sigma) \upharpoonleft_{\mathrm{eq}} \left((\Sigma -) \square - \vdash L_{\Sigma\text{-}\mathbf{Mon}} = R_{\Sigma\text{-}\mathbf{Mon}}\right) \tag{14}$$

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

$$x : \Sigma\tau, y : \tau \vdash \mu(\mathrm{op}\ x, y) = \mathrm{op}((\Sigma\mu)(\theta_{\tau,\langle \tau,\eta \rangle}(x, y))) : \tau \tag{15}$$

The equation (15) expresses that the operation op commutes with monoid multiplication. When using the monoidal category $\langle \mathbf{Endo}_f(\mathbf{Set}), \bullet, V \rangle$ for modelling higher-order abstract syntax, which was the original context where Fiore et al. [1999] introduced Σ-monoids, this equation expresses the sensible condition that operations must commute with substitution. However, this equation might not be desirable in other contexts; for example, when using $\langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ to model computational effects, this equation expresses that effectful operations must commute with sequential composition, which is not true in general. Those effectful operations that do satisfy this condition and have signature $\Sigma = A \square -$ for some $A \in \mathscr{E}$ are called *algebraic operations* [Jaskelioff and Moggi 2010; Plotkin and Power 2001]. We will say more about the equation (15) shortly in Example 3.7 and see that imposing it on Σ-monoids actually does not lose generality.

A model of the equational system Σ-**Mon** is called a Σ-monoid. When the monoidal category $\mathscr{E}$ is cocomplete and functors Σ, Γ, $\square$ all preserve colimits of α-chains for some limit ordinal α, Theorem 3.3 ensures the existence of free Σ-monoids. When $\mathscr{E}$ is additionally closed, such as $\langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ in §2.1, there is again a simple description of the free Σ-monoid [Fiore and Hur 2007; Fiore and Saville 2017]: it is carried by the initial algebra $\mu X.I + A \square X + \Sigma X$. This formula has many applications in modelling abstract syntax: variable binding [Fiore and Szamozvancev 2022], explicit substitution [Ghani et al. 2006], and scoped operations [Piróg et al. 2018].

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:

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

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

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

$$(\coprod_E 1) \circ M = \coprod_E (1 \circ M) = \coprod_E 1 \quad \longrightarrow \quad M \tag{16}$$

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

$$(\coprod_E 1) \star F \cong \coprod_E (1 \star F) = \coprod_E (\textstyle\int^{a,b} 1a \times Fb \times -^{a \times b}) \cong \coprod_E (\int^b Fb) \quad \to \quad F \tag{17}$$

Note that exception throwing for monads (16) and for applicatives (17) differ by the domain 1 vs $\int^b Fb$. This reflects the nature of applicative functors that computations are independent, so the computation after exception throwing is not necessarily discarded.

Example 3.7 (Exception Catching)**.** Although exception *throwing* is an algebraic operation, it is well known that *catching* is not: if we were to model it as an algebraic operation $catch : M \times M \to M$ on a monad M such that $catch\ \langle p, h \rangle$ means catching exceptions possibly thrown by p and handling exceptions using h, then the equation (15) for $\Sigma M = M \times M$ implies that

$$ph : M \times M, k : M \vdash \mu(catch\ ph, k) = catch\ \langle \mu(\pi_1\ ph,\ k), \mu(\pi_2\ ph,\ k) \rangle : M \tag{18}$$

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

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

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

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

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

where the boxed Y is inserted using $f : I \to Y$. The intuition for the signature Σ_{Ec} is that the first operation $1 \circ M \to M$ is *throw*ing an exception as in Example 3.6, and the second operation

$$catch : (\mathrm{Id} \times \mathrm{Id}) \circ M \circ M \cong (M \times M) \circ M \quad \longrightarrow \quad M \tag{19}$$

is catching. The trick here to avoid the undesirable equation (18) is that *catch* has after $M \times M$ an additional $- \circ M$ that represents an *explicit continuation* after the scoped operation *catch* [Piróg et al. 2018]: $catch\ (\langle p, h \rangle, k)$ is understood as handling the exception in p with h and then continuing as k. Then the equation (15) of Σ-monoids instantiates to

$$ph : M \times M, k : M, k' : M \vdash \mu(catch\ (ph, k), k') = catch\ (ph, \mu(k, k')) : M. \tag{20}$$

Unlike (18), 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.

This trick applies more generally: for all functors $\Phi : \mathscr{E} \to \mathscr{E}$ and monoids M in a monoidal category $\mathscr{E}$, define $\Sigma = (\Phi -) \square -$ and a pointed strength for Σ:

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

Then arrows $f : \Phi M \to M$ *without any condition* are in bijection with arrows $g : (\Phi M) \square M \to M$ that satisfy the equation (15) of Σ-monoids instantiated with the Σ:

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

Therefore, imposing (15) on Σ-monoids does not lose generality.

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

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

where $\eta : \mathsf{I} \to \tau$, $\mathit{throw} : 1 \to \tau$, and $\mathit{catch} : (\tau \times \tau) \square \tau \to \tau$. These equations can be alternatively presented with an empty context by replacing all the k's with η as in $\cdot \vdash \mathit{catch}(\langle \mathit{throw}, \eta \rangle, \eta) = \eta : \tau$, which is equivalent to the first equation above, since by (20), $\mathit{catch}(\langle x, y \rangle, \eta); k = \mathit{catch}(\langle x, y \rangle, k)$.

Example 3.8. Let S be a set. The theory ST_S of *monads with global S-state* [Plotkin and Power 2002] can be generally defined for monoids as follows. The theory ST_S is Σ_{ST_S}-**Mon** with signature Σ_{ST_S} denoted by $((\prod_S \mathsf{I}) \square \tau) + ((\coprod_S \mathsf{I}) \square \tau)$, whose first component represents an operation $g : (\prod_S \mathsf{I}) \square \tau \to \tau$ reading the state, and the second component represents an operation $p : (\coprod_S \mathsf{I}) \square \tau \to \tau$ 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

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

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

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

3.4 Functorial Translations

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

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

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

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

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

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

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

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

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

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

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

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

Colimits. Colimits in $\mathbf{Eqs}(\mathscr{C})$ allow one to 'glue' equational systems. For example, the equational system for rings can be obtained by first taking the coproduct of **Grp** and **Mon** and then taking a suitable coequaliser $L \rightrightarrows$ **Grp** + **Mon** encoding the interaction of operations.

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

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

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

Lemma 3.14. *Let $\dot{\Sigma} \in \mathbf{Eqs}(\mathscr{C})$ for $\mathscr{C}$ a category with finite coproducts, and for $i \in \{1, 2\}$, let $\Theta_i : \mathscr{C} \to \mathscr{C}$ be a functorial signature and $E_i = (\Theta_i \vdash L_i = R_i)$ be an equation. Let T_1 and T_2 in the diagram below be the inclusion translations, then the following is a pushout diagram of T_1 and T_2:*

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

where $E = (\Theta_1{+}\Theta_2 \vdash [L_1 \circ \alpha_1, L_2 \circ \alpha_2] = [R_1 \circ \alpha_1, R_2 \circ \alpha_2])$ *and* $\alpha_i : (\Sigma{+}(\Phi_1{+}\Phi_2))\text{-}\mathbf{Alg} \to (\Sigma{+}\Phi_i)\text{-}\mathbf{Alg}$ *is the projection functor.*

4 MONOIDAL THEORY FAMILIES

Motivated by the principle *notions of computations as monoids* (§2), we are primarily interested in equational theories that extend the theory **Mon** of monoids with more operations. The more precise way to say it now is that we are interested in the coslice category $\mathbf{Mon}/\mathbf{Eqs}_c(\mathscr{E})$. Moreover, sometimes we are only interested in operations of some special forms, e.g. variable-binding operations, instead of all possible operations on monoids. Thus in this section, we do a finer classification of theories in $\mathbf{Mon}/\mathbf{Eqs}_c(\mathscr{E})$, grouping them into different *monoidal theory families*. It turns out that the simplest kind of operations, the algebraic ones, play a special role among all operations.

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

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

Definition 4.2. A monoidal category $\mathscr{E}$ is called *cordial* when it is cocomplete and its monoidal product $\Box : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ is right distributive (i.e. $\Box$ preserves all coproducts in the first argument) and preserves colimits of α-chains for some limit ordinal α.

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

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

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

where $T : \mathbf{Mon} \hookrightarrow \Sigma\text{-}\mathbf{Mon} \,\lceil_{\mathrm{eq}}\, E$ is the inclusion translation, and E is an arbitrary functorial equation whose context is a *constant functor* $\mathbf{K}_B X = B$ for all X. In other words, $\mathbf{ALG}(\mathscr{E})$ contains all equational systems $\Sigma\text{-}\mathbf{Mon}$ (14) extended with an equation for some $\Sigma = A \Box -$ with the associator of $\mathscr{E}$ as the pointed strength $\alpha_{A,X,Y} : (A \,\Box\, X) \,\Box\, Y \cong A \,\Box\, (X \,\Box\, Y)$.

The category $\mathbf{ALG}(\mathscr{E})$ satisfies the conditions in Definition 4.1 since it is closed under coproducts following Lemma 3.14, and all equational systems (21) in $\mathbf{ALG}(\mathscr{E})$ have free algebras by Theorem 3.3, since the signature and context functor of (21) preserve colimits of α-chains for some ordinal α.

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

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

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

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

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

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

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

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

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

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

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

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

$$\{\langle \Sigma\text{-}\mathbf{Mon} \upharpoonleft_{\mathrm{eq}} E,\ T \rangle \mid A, B, C \in \mathscr{E},\ \Sigma = (A \,\square\, - \,\square\, -) + (B \,\square\, -),\ E = (\mathbf{K}_C \vdash L = R)\} \tag{22}$$

where $T : \mathbf{Mon} \to \Sigma\text{-}\mathbf{Mon} \upharpoonleft_{\mathrm{eq}} E$ is the inclusion translation and $\mathbf{K}_C : \mathscr{E} \to \mathscr{E}$ is the constant functor mapping to $C \in \mathscr{E}$. The pointed strength $\theta_{X,\langle Y,f \rangle}$ of Σ needed in the definition of Σ-**Mon** (14) is as follows, where the boxed Y is inserted using $f : I \to Y$:

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

Piróg et al. [2018] introduced scoped operations to model non-algebraic operations that delimit scopes. As explained in Example 3.7, the trick is to let the operation take an explicit continuation.

A direct corollary of Theorem 4.4 is that the initial-algebra preserving coreflection there restricts to $\mathbf{ALG}(\mathscr{E}) \overset{\top}{\hookleftarrow} \mathbf{SCP}(\mathscr{E})$. Thus the terms of scoped operations can be alternatively expressed with only algebraic ones, but as argued by Piróg et al. [2018] and Yang et al. [2022], the models of scoped operations are different from those of the coreflected algebraic operations.

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

A *binding signature* $\langle O, a \rangle$ consists of a set O of operations and an arity assignment $a : O \to \mathbb{N}^*$ of a sequence of natural numbers to each operation. Each $o \in O$ with $a(o) = \langle n_i \rangle_{1 \leqslant i \leqslant k}$ stands for an operation taking k arguments, each binding n_i variables. For example, the binding signature for untyped λ-calculus has two operations $\{app, abs\}$: application $a(app) = \langle 0, 0 \rangle$ has two arguments, each binding no variables; abstraction $a(abs) = \langle 1 \rangle$ has one argument that binds one variable.

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

$$\Sigma_{\langle O,a \rangle} = \coprod_{o \in O,\ a(o) = \langle n_i \rangle_{1 \leqslant i \leqslant k}} \prod_{1 \leqslant i \leqslant k} (-)^{V^{n_i}} \tag{23}$$

where $(-)^{V^{n_i}}$ is the exponential by n_i-fold product of the monoidal unit V. It has a pointed strength:

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

where $t_{o,i}$ is the adjunct of $(X^{V^{n_i}} \bullet Y) \times V^{n_i} \xrightarrow{\mathrm{id} \times \eta^Y} (X^{V^{n_i}} \bullet Y) \times (V^{n_i} \bullet Y) \cong (X^{V^{n_i}} \times V^{n_i}) \bullet Y \to X \bullet Y$.

The monoidal theory family $\mathbf{VAR}(\mathbf{Set}^{\mathbf{Fin}}) \subseteq \mathbf{Mon}/\mathbf{Eqs}(\mathbf{Set}^{\mathbf{Fin}})$ then contains objects

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

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

Again, the coreflector in Theorem 4.4 allows us to turn every theory in $\mathbf{VAR}$ into one with only algebraic operations but has isomorphic initial algebras. For example, under the coreflection, the theory Λ of untyped λ-calculus is turned into a theory $\lfloor \Lambda \rfloor$ which has an ordinary n-ary operation t for *every* λ-term t with n free variables, together with suitable equations. The equational systems Λ and $\lfloor \Lambda \rfloor$ have isomorphic initial algebras (as monoids).

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

5 MODULAR MODELS OF MONOIDS

Now we come to the second part of this paper on *semantic* modularity. The idea is simple: given a theory $\ddot{\Psi}$ in a monoidal theory family $\mathcal{F}$, we would like to consider *modular models* M of $\ddot{\Psi}$-operations that can coexist with any other theories $\ddot{\Sigma} \in \mathcal{F}$, in the sense that M can transform *every* model of *every* theory $\ddot{\Sigma} \in \mathcal{F}$ to a model of $\ddot{\Sigma} + \ddot{\Psi}$ (§5.1). We have two equivalent formulations, one based on $\mathbf{CAT}$-*valued functors* (Definition 5.2), and another based on *fibrations* (Theorem 5.4). We also explain how modular models are used to interpret abstract syntax in a modular way (§5.2).

5.1 Modular Models

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

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

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

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

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

We show the fibrational formulation before diving into any examples of modular models. We will only need the very basics about fibrations (see e.g. Jacobs [1999, Chapter 1]). For every monoidal theory family $\mathcal{F}$, the **CAT**-valued functor $(-)$-**Alg** : $\mathcal{F}^{op} \to \mathbf{CAT}$ 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 equational systems* in $\mathcal{F}$.

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

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

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

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

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

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

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

$$\begin{array}{ccc} (\mathcal{F} + \ddot{\Psi})\text{-}\mathbf{Alg} & \xrightarrow{\Xi} & \mathcal{F}\text{-}\mathbf{Alg} \\ {\scriptstyle Q}\downarrow & \lrcorner & \downarrow{\scriptstyle P} \\ \mathcal{F} & \xrightarrow[-+\ddot{\Psi}]{} & \mathcal{F} \end{array} \tag{24}$$

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

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

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

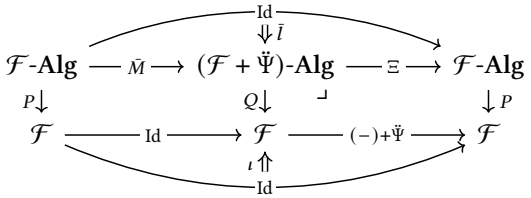
The pullback (24) also induces two functors Q and Ξ. The functor $Q : (\mathcal{F} + \ddot\Psi)\text{-}\mathbf{Mon} \to \mathcal{F}$ is the projection to $\mathcal{F}$, and it is a fibration with a split cleavage similar to that of P. The functor $\Xi : (\mathcal{F}+\dot\Psi)\text{-}\mathbf{Mon} \to \mathcal{F}\text{-}\mathbf{Mon}$ maps objects $\langle \dot\Sigma, T_\Sigma, A, \alpha, \beta \rangle$ to $\langle \langle \dot\Sigma, T_\Sigma \rangle + \langle \dot\Psi, T_\Psi \rangle, A, [\alpha, \beta] \rangle$. Additionally, the pair $\langle \Xi, - + \dot\Psi \rangle$ is a morphism of (split) fibrations from Q to P.

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

Theorem 5.4. *Modular models M of some $\ddot\Psi \in \mathcal{F}$ as in Definition 5.2 are in bijection with functors $\bar M : \mathcal{F}\text{-}\mathbf{Alg} \to (\mathcal{F} + \ddot\Psi)\text{-}\mathbf{Alg}$ such that $Q \circ \bar M = P$ with P and Q as in (24). A modular model M is strong (resp. strict) iff $\bar M$ is a morphism of fibrations (resp. split fibrations) from P to Q.*

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

Definition 5.5. A *lifting* $\bar l$ for a modular model $\bar M$ is a natural transformation $\bar l : \mathrm{Id} \to \Xi \circ \bar M$ such that $P \circ \bar l = \iota \circ P$ where $\iota : \mathrm{Id} \to - + \ddot\Psi$ is the coprojection in $\mathcal{F}$ and P, Q, Ξ are as in (24):



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

$$\ddot\Sigma \mapsto \langle \dot\Sigma,\ T_\Sigma,\ \mu\dot\Sigma, \alpha^\Sigma : \Sigma(\mu\dot\Sigma) \to \mu\dot\Sigma \rangle \quad (T : \ddot\Sigma \to \ddot\Psi) \mapsto \langle T,\ u : \langle \mu\dot\Sigma,\ \alpha^\Sigma \rangle \to T\langle \mu\dot\Psi,\ \alpha^\Psi \rangle \rangle \qquad (25)$$

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

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

Example 5.7. For a concrete example, let $\mathscr{E}$ be $\langle \mathbf{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ for lκp $\mathscr{C}$. A strict modular model M for the theory ET_E of *exception throwing* (Example 3.6) in the family $\mathbf{ALG}(\mathscr{E})$ of algebraic operations is given by a family of functors $M_{\ddot\Sigma} : \ddot\Sigma\text{-}\mathbf{Alg} \to (\ddot\Sigma + \mathrm{ET}_E)\text{-}\mathbf{Alg}$ natural in $\ddot\Sigma \in \mathbf{ALG}(\mathscr{E})$. Recall that objects of $\ddot\Sigma\text{-}\mathbf{Alg}$ 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 $M_{\ddot\Sigma}$ to a $(\ddot\Sigma + \mathrm{ET}_E)$-algebra carried by the *exception monad transformer* $C_A = A \circ (\mathbb{E} + \mathrm{Id})$, where $\mathbb{E}$ is the E-fold product of 1 in $\mathbf{Endo}_\kappa(\mathscr{C})$. The carrier is equipped with operations $[\alpha^\sharp, \beta] :\ (\Sigma \circ C_A) + \mathbb{E} \to C_A$, 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[2]:

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

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

5.2 Interpretation with Modular Models

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

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

$$h_{\ddot\Sigma} : \mu(\ddot\Sigma + \ddot\Psi) \to M_{\ddot\Sigma} \langle \mu\dot\Sigma, \alpha^\Sigma \rangle. \tag{26}$$

The intuition is that this morphism *modularly* interprets the $\dot\Psi$-operations in the abstract syntax $\mu(\ddot\Sigma + \ddot\Psi)$ with a modular model M, leaving operations from the other theory $\ddot\Sigma$ uninterpreted. For example, with the modular model in Example 5.7, we can interpret terms of exception throwing mixed with other algebraic operations: (26) specialises to $h_{\ddot\Sigma} : \mu(\ddot\Sigma + \mathrm{E}\tau_E) \to (\mu\dot\Sigma \circ (\mathbb{E} + \mathrm{Id}))$.

As a special case, let $\ddot\Sigma$ be the theory $\langle \mathbf{Mon}, \mathrm{id} \rangle$ of monoids (which is always in $\mathcal{F}$ since it is the initial object of $\mathcal{F}$ and $\mathcal{F}$ is closed under finite coproducts), and then $\mu\dot\Sigma$ is the initial monoid I and $\mu(\ddot\Sigma + \ddot\Psi) \cong \mu\dot\Psi$. In this case, the morphism $h_{\ddot\Sigma} : \mu\dot\Psi \to \ddot M \langle I, \alpha^I \rangle$ (26) interprets the abstract syntax $\mu\dot\Psi$ without anymore uninterpreted operations.

The interpretation (26) can be formulated as a natural transformation in $\ddot\Sigma$ by using the functor $(-)^\star : \mathcal{F} \to \mathcal{F}\text{-}\mathbf{Alg}$ (25) sending every $\ddot\Sigma \in \mathcal{F}$ to its initial algebra in $\mathcal{F}\text{-}\mathbf{Alg}$.

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

$$\dot h^M : (- + \ddot\Psi)^\star \to \Xi \bar M (-)^\star : \mathcal{F} \to \mathcal{F}\text{-}\mathbf{Alg} \tag{27}$$

such that all $\dot h^M_{\ddot\Sigma} = \langle \mathrm{id}_{\ddot\Sigma + \ddot\Psi}, h_{\ddot\Sigma} \rangle$, where $\bar M : \mathcal{F}\text{-}\mathbf{Alg} \to (\mathcal{F} + \ddot\Psi)\text{-}\mathbf{Alg}$ is the equivalent form of M in Theorem 5.4, and $\Xi : (\mathcal{F} + \ddot\Psi)\text{-}\mathbf{Alg} \to \mathcal{F}\text{-}\mathbf{Alg}$ is defined as in (24).

Remark. Since $\ddot\Sigma + \ddot\Psi$ extends the theory of monoids, the interpretation (27) is always a monoid morphism, and thus it preserves monoid multiplication μ. This is called the *semantic substitution lemma* [Tennent 1991] for $\mathscr{E} = \langle \mathbf{Set}^{\mathbf{Fin}}, \bullet, V \rangle$ since μ stands for substitution in this case.

Liftings of modular models (Definition 5.5) are useful for *reusing existing interpretations*. Suppose that some operations are first modelled by a theory $\ddot\Sigma$ and interpreted with some model $A \in \ddot\Sigma\text{-}\mathbf{Alg}$,

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

and later some new operations $\ddot{\Psi}$ are added, and $\ddot{\Psi}$ has a modular model M with a lifting l. Then by the initiality of $\mu\ddot{\Sigma}$, the following diagram commutes:

$$\begin{array}{lcc} \text{Syntax} & \mu\ddot{\Sigma} \xrightarrow{\mu\iota_1} \mu(\ddot{\Sigma} + \ddot{\Psi}) & \\ & {\scriptstyle u_1}\downarrow \quad\quad \downarrow{\scriptstyle u_2} & \text{in } \ddot{\Sigma}\text{-}\mathbf{Alg} \\ \text{Semantics} & A \xrightarrow{l_A} M_{\ddot{\Sigma}}A & \end{array}$$

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

6 CONSTRUCTIONS OF MODULAR MODELS

The definition of modular models is just a mathematical formulation of the idea of semantic modularity, and certainly, what is more interesting is the concrete examples and constructions of modular models. In this section we show several constructions and examples, which hopefully evidence that our framework is not a vacuous abstraction.

6.1 Modular Models from Monoid Transformers

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

Theorem 6.1. *Let $\mathcal{T} = \mathbf{ALG}(\mathscr{E})$ for a cordial monoidal category $\mathscr{E}$. For each $\ddot{\Psi} = \langle \dot{\Psi}, T_\Psi \rangle \in \mathcal{T}$, a functor $H : \mathbf{Mon}(\mathscr{E}) \to \dot{\Psi}\text{-}\mathbf{Alg}$ and a natural transformation $\tau : \mathrm{Id} \to T_\Psi \circ H$ as depicted in the diagram on the left below can be extended to a strict modular model $\bar{M}$ of $\ddot{\Psi}$ such that the diagram on the right below commutes, and $\bar{M}$ has a lifting $\bar{l}_{\langle \langle \dot{\Sigma}, T_\Sigma \rangle, A, \alpha \rangle} = \tau_{\langle A, T_\Sigma \alpha \rangle}$:*

$$\begin{array}{ccc} & \dot{\Psi}\text{-}\mathbf{Alg} & \\ {\scriptstyle H}\nearrow & \Uparrow{\scriptstyle \tau} & \searrow{\scriptstyle T_\Psi} \\ \mathbf{Mon}(\mathscr{E}) & \xrightarrow[\mathrm{Id}]{} & \mathbf{Mon}(\mathscr{E}) \end{array} \qquad \begin{array}{ccc} \mathcal{F}\text{-}\mathbf{Alg} & \xrightarrow{\bar{M}} & (\mathcal{F} + \ddot{\Psi})\text{-}\mathbf{Alg} \\ \downarrow & & \downarrow \\ \mathbf{Mon}(\mathscr{E}) & \xrightarrow[H]{} & \dot{\Psi}\text{-}\mathbf{Alg} \end{array} \tag{28}$$

where the unlabelled vertical arrows are the evident projection functors.

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

Example 5.7 is in fact this theorem applied to the exception monad transformer. The *state monad transformer* $A \mapsto (A(S \times -))^S$ for a set S with $|S| < \kappa$ together with its model for the theory ST_S of *mutable state* (Example 3.8) yields a modular model of ST_S in $\mathbf{ALG}(\mathbf{Endo}_\kappa(\mathscr{C}))$. The *list monad transformer* $A \mapsto \mu X.A(1 + (- \times X))$ [Jaskelioff and Moggi 2010] with its model for the theory of *explicit nondeterminism* also gives rise to a modular model. Moreover, it allows us to obtain modular models of theories in $\mathbf{ALG}(\mathscr{E})$ from ordinary models by taking coproducts of monoids.

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

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

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

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

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

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

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

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

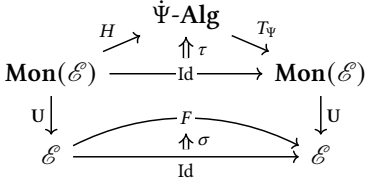
Then catch is $s \circ C_A$ followed by the arrow denoted by the following term:

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

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

Theorem 6.4. *For each* $\langle \dot{\Psi}, T_\Psi \rangle \in \mathbf{SCP}(\mathscr{E})$ *over a cordial and* closed *monoidal category* $\mathscr{E}$*, a functor* $H : \mathbf{Mon}(\mathscr{E}) \to \dot{\Psi}\text{-}\mathbf{Alg}$ *and a natural transformation* $\tau : \mathrm{Id} \to T_\Psi \circ H$ *such that there is some* $F : \mathscr{E} \to \mathscr{E}$ *and* $\sigma : \mathrm{Id} \to F$ *satisfying* $\mathbf{U} \circ T_\Psi \circ H = F \circ \mathbf{U}$ *and* $\tau \circ \mathbf{U} = \sigma \circ U$ *can be extended to a strict modular model* M *of* $\langle \dot{\Psi}, T_\Psi \rangle$ *with a lifting* l *such that (28) commutes and* $l_{\langle \langle \dot{\Sigma}, T_\Sigma \rangle,\ A,\ \alpha \rangle} = \tau_{\langle A,\ T_\Sigma \alpha \rangle}$.



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

6.2 Free Modular Models

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

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

translation; **Grp** with the inclusion translation $T : \mathbf{Mon} \to \mathbf{Grp}$ (Example 3.10); and the theory **BLat** of bounded lattices with the translation that maps monoid multiplication to lattice join $\vee$, monoid identity to lattice bottom $\bot$. The following diagram in **CAT** does not commute:

$$\begin{array}{ccc} \mathbf{Mon}\text{-}\mathbf{Alg} & \xleftarrow{\quad T \quad} & \mathbf{Grp}\text{-}\mathbf{Alg} \\ {\scriptstyle \mathbf{F}_{\mathbf{Mon}}}\downarrow & & \downarrow{\scriptstyle \mathbf{F}_{\mathbf{Grp}}} \\ (\mathbf{Mon}+\mathbf{BLat})\text{-}\mathbf{Alg} & \xleftarrow{\quad T+\mathbf{BLat} \quad} & (\mathbf{Grp}+\mathbf{BLat})\text{-}\mathbf{Alg} \end{array}$$

The theory $\mathbf{Mon}+\mathbf{BLat}$, bounded lattices whose join $\vee$ and $\bot$ form a monoid, is isomorphic to **BLat** since $\langle \vee, \bot \rangle$ of a lattice is already a monoid, so $\mathbf{F}_{\mathbf{Mon}}(TG)$ for every group G is the free bounded lattice over G as a monoid. On the other hand, the theory $\mathbf{Grp}+\mathbf{BLat}$, bounded lattices whose $\langle \vee, \bot \rangle$ form a group, has only trivial models since for every element x, $\bot = x \vee x^{-1}$, and by the idempotent law of join, $\bot = (x \vee x) \vee x^{-1} = x \vee (x \vee x^{-1}) = x \vee \bot = x$. Therefore $(T+\mathbf{BLat})(F_{\mathbf{Grp}}G)$ is always the trivial bounded lattice for every group G. However, the diagram above can be made an oplax transformation, resulting in a non-strict modular model of **BLat**.

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

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

$$\begin{array}{ccc} \ddot{\Sigma}\text{-}\mathbf{Alg} & \xleftarrow{\quad T \quad} & \ddot{\Sigma}'\text{-}\mathbf{Alg} \\ {\scriptstyle \mathbf{U}_{\ddot{\Sigma}}}\uparrow\ \downarrow{\scriptstyle \mathbf{F}_{\ddot{\Sigma}}} & & {\scriptstyle \mathbf{U}_{\ddot{\Sigma}'}}\uparrow\ \downarrow{\scriptstyle \mathbf{F}_{\ddot{\Sigma}'}} \\ (\ddot{\Sigma}+\ddot{\Psi})\text{-}\mathbf{Alg} & \xleftarrow[T+\ddot{\Psi}]{} & (\ddot{\Sigma}+\ddot{\Psi})\text{-}\mathbf{Alg} \end{array}$$

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

Example 6.6. The free modular model is remarkably general as it works for $\mathbf{Mon}/\mathbf{Eqs}_c(\mathscr{E})$. For an example, let us consider how the free modular model adds new operations to a model of λ-calculus. Let $\mathscr{E}$ be $\langle \mathbf{Set}^{\mathbf{Fin}}, \bullet, V \rangle$ (5). As we mentioned earlier (23), the syntax of λ-calculus can be presented as an equational system $\Lambda = \Sigma_{\langle O,a \rangle}\text{-}\mathbf{Mon}$ for $O = \{ app, abs \}$ with $a(abs) = \langle 1 \rangle$ and $a(app) = \langle 0, 0 \rangle$. Models of Λ can be obtained from reflexive objects $U \cong U^U$ in any cartesian closed category C: every U induces a functor $\bar{U} : \mathbf{Fin} \to \mathbf{Set}$ with $n \mapsto \mathscr{C}(U^n, U)$. The functor $\bar{U}$ has a monoid structure $[\eta_U, \mu_U]$ (similar to that of the continuation monad), and it is a model of Λ [Hyland 2017]:

$$abs_U : \bar{U}^V n \cong \mathscr{C}(U^{n+1}, U) \cong \mathscr{C}(U^n, U^U) \cong \mathscr{C}(U^n, U) \cong \bar{U}n$$

$$app_U : (\bar{U} \times \bar{U})n \cong \mathscr{C}(U^n, U \times U) \cong \mathscr{C}(U^n, U^U \times U) \xrightarrow{eval_{U^U} \cdot -} \mathscr{C}(U^n, U) \cong \bar{U}n$$

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

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

quotiented by equations of Λ and ST_S, as well as equations saying that the Λ operations of the initial algebra acting on the first component $\bar{U}$ is the same as the model $[\eta_U, \mu_U, abs_U, app_U]$ of $\bar{U}$.

6.3 Composition and Fusion of Modular Models

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

Definition 6.7. Given (strict/strong) modular models M of $\ddot{\Psi} \in \mathcal{F}$ and N of $\ddot{\Phi} \in \mathcal{F}$ in the functor form as in Theorem 5.4, their *composite* $M \triangleright N$ is a (strict/strong) modular model of $\ddot{\Psi} + \ddot{\Phi}$:

$$M \triangleright N : \mathcal{F}\text{-}\mathbf{Alg} \to (\mathcal{F} + \ddot{\Psi} + \ddot{\Phi})\text{-}\mathbf{Alg}$$

defined by the pullback property (24) for $(\mathcal{F} + \ddot{\Psi} + \ddot{\Phi})$-**Alg** and the commutativity of the diagram:

$$\begin{array}{ccccccccc}
\mathcal{F}\text{-}\mathbf{Alg} & \xrightarrow{M} & (\mathcal{F}+\ddot{\Psi})\text{-}\mathbf{Alg} & \xrightarrow{\Xi} & \mathcal{F}\text{-}\mathbf{Alg} & \xrightarrow{N} & (\mathcal{F}+\ddot{\Phi})\text{-}\mathbf{Alg} & \xrightarrow{\Xi} & \mathcal{F}\text{-}\mathbf{Alg} \\
{\scriptstyle P}\downarrow & & {\scriptstyle Q}\downarrow & & {\scriptstyle P}\downarrow & & {\scriptstyle Q'}\downarrow & & {\scriptstyle P}\downarrow \\
\mathcal{F} & \xrightarrow[\mathrm{Id}]{} & \mathcal{F} & \xrightarrow[-+\ddot{\Psi}]{} & \mathcal{F} & \xrightarrow[\mathrm{Id}]{} & \mathcal{F} & \xrightarrow[-+\ddot{\Phi}]{} & \mathcal{F}
\end{array}$$

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

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

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

A composite model $M \triangleright N$ can be used for interpreting $(\ddot{\Sigma} + (\ddot{\Psi} + \ddot{\Phi}))^\star$ using $\dot{h}^{M \triangleright N}$ (27). Since $(\ddot{\Sigma} + (\ddot{\Psi} + \ddot{\Phi}))^\star \cong ((\ddot{\Sigma} + \ddot{\Psi}) + \ddot{\Phi})^\star$, 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 $(\ddot{\Sigma} + (\ddot{\Psi} + \ddot{\Phi}))^\star$.

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

$$\begin{array}{ccc}
(\ddot{\Sigma} + (\ddot{\Psi} + \ddot{\Phi}))^\star & \xrightarrow{\cong} & ((\ddot{\Sigma} + \ddot{\Psi}) + \ddot{\Phi})^\star \\
{\scriptstyle \dot{h}^{M \triangleright N}_{\ddot{\Sigma}}}\downarrow & & \downarrow{\scriptstyle \dot{h}^{N}_{\ddot{\Sigma}+\ddot{\Psi}}} \\
\Xi N \Xi M \ddot{\Sigma}^\star & \xleftarrow{\Xi N \dot{h}^{M}_{\ddot{\Sigma}}} & \Xi N (\ddot{\Sigma} + \ddot{\Psi})^\star
\end{array}$$

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

6.4 Modular Models in Symmetric Monoidal Categories

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

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

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

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

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

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

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

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

There is another way to compose two applicatives, namely $A \circ B$ [Mcbride and Paterson 2008]. In this way, the B-computation can *depend* on the result of A.

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

Example 6.12. To highlight the difference between Theorem 6.10 and Theorem 6.11, let $\dot{A}$ 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 $throw : \mathbb{E} \star (\mathbb{E} + \mathrm{Id}) \to (\mathbb{E} + \mathrm{Id})$. Using Theorem 6.11, it can be extended to a modular model using $(\mathbb{E} + \mathrm{Id}) \circ B \cong (\mathbb{E} + B)$ for all applicatives B. In this model, it holds that for all elements $x, y \in (\mathbb{E} + B)X$, $throw\ \langle e, x\rangle = inj_1\ e = throw\ \langle e, y\rangle$, 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 6.10.

Phasing. As our final example, we show an interesting modular model for *multi-phase computation*, generalising the construction that Kidney and Wu [2021] introduce for *breadth-first search*. The theory $\textsc{Pha} \in \mathbf{SCP}(\mathscr{E})$ has a unary scoped operation $later : A \square A \cong I \square A \square A \to A$. The intuition is that a program may have multiple phases of execution, and the operation $later\ \langle p, k\rangle$ delays the execution of p by a phase, 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 phase 1, p_{22} and p_{21} at phase 2, and p_3 at phase 3.

Given a monoid $\dot{A} = \langle A, \mu^A, \eta^A\rangle$ in a symmetric closed cordial 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 $\eta^{S_A} = [\![\cdot \vdash out^\circ\ (\mathsf{inj}_2 *)]\!]$ and μ^{S_A} being

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

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 phase, and μ^{S_A} merges two lists by multiplying computations at the same phase. The *later* operation on S_A is defined as $[\![(p : S_A, k : S_A \vdash \mu^{S_A}(out^\circ(\mathsf{inj}_1(\eta^A, p), k)) : S_A]\!]$.

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

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

REFERENCES

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

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

Robert Atkey. 2009. Parameterised notions of computation. *Journal of Functional Programming* 19, 3–4 (2009), 335–376. https://doi.org/10.1017/S095679680900728X

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

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

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

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

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

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

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

Marcelo Fiore and Chung-Kil Hur. 2007. Equational Systems and Free Constructions. In *Proceedings of the 34th International Conference on Automata, Languages and Programming* (Wrocław, Poland) *(ICALP'07)*. Springer-Verlag, Berlin, Heidelberg, 607–618. https://doi.org/10.1007/978-3-540-73420-8_53

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

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

Marcelo Fiore, Gordon Plotkin, and Daniele Turi. 1999. Abstract Syntax and Variable Binding. In *Proceedings. 14th Symposium on Logic in Computer Science*. 193–202. https://doi.org/10.1109/LICS.1999.782615

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

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

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. *Higher-Order and Symbolic Computation* (2006). https://doi.org/10.1007/s10990-006-8748-4

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

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

Claudio Hermida. 2000. Representable Multicategories. *Advances in Mathematics* 151, 2 (2000), 164–225. https://doi.org/10.1006/aima.1999.1877

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/10.1007/978-3-642-31113-0_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. 2017. Classical Lambda Calculus in Modern Dress. *Mathematical Structures in Computer Science* 27, 5 (2017), 762–781. https://doi.org/10.1017/S0960129515000377

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

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

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

G.M. Kelly and John 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

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

Anders Kock. 1972. Strong Functors and Monoidal Monads. *Archiv der Mathematik* 23 (12 1972), 113–120. https://doi.org/10.1007/BF01304852

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

J. Lambek and P. J. Scott. 1986. *Introduction to Higher Order Categorical Logic.* Cambridge University Press.

F. William Lawvere. 1963. Functorial Semantics of Algebraic Theories. *Proceedings of the National Academy of Sciences* 50, 5 (1963), 869–872. https://doi.org/10.1073/pnas.50.5.869

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

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

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

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

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

Dylan McDermott and Tarmo Uustalu. 2022b. What Makes a Strong Monad? *Electronic Proceedings in Theoretical Computer Science* 360 (Jun 2022), 113–133. https://doi.org/10.4204/EPTCS.360.6

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

Eugenio Moggi. 1989b. Computational Lambda-Calculus and Monads. In *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.).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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