Modular Models of Monoids with Operations

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

Abstract

We develop a theory of *modular models* of equational theories of monoids equipped with additional operations. Modular models of different theories in a family can be composed to form models of the coproduct of the theories, making both abstract syntax and semantic models composable in a modular way. As examples, we show how modular models of algebraic operations and scoped operations on monoids can be constructed from monoid transformers.

1 Introduction

Equational theories, also known as algebraic theories, of monoids with additional operations, called Σ-*monoids* by Fiore et al. [1999], are widely used in the study of programming languages. *Abstract syntax* is the initial algebra of Σ-monoids, and it is interpreted by other *semantic algebras* (*models*) of Σ-monoids following the initiality of the syntax. Important applications in various categories include:

(i) In the category of sets, free monoids, i.e. *lists*, are the fundamental data structure in functional programming, and their universal properties underpin the study of list-processing functions by the Bird–Meertens formalism [Bird and de Moor 1997; Bird 1987; Meertens 1986]. Free monoids with additional operations are *trees*, whose monoid unit is the empty tree, and multiplication is substitution of leaf nodes.

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

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

(iv) In a presheaf category, free monoids with additional operations are used to model *higher-order abstract syntax* with variable bindings [Fiore and Hur 2010; Fiore and Szamozvancev 2022; Fiore et al. 1999; Fiore and Turi 2001], for which the monoid multiplication means substitution of variables.

This approach is both theoretically and practically appealing for various reasons: (i) interpretation is compositional automatically; (ii) syntactic structures can be interpreted with different semantic models; (iii) equational theories can be combined [Hyland et al. 2006], allowing syntax to be specified in a modular way, known as *data types à la carte* in functional programming [Swierstra 2008].

Lack of Modularity for Models. There is a caveat: although modularity of syntax is attained by combining equational theories, models are *not* modular, in the sense that there is no canonical ways to combine two models of theories T_1 and T_2 respectively—not necessarily initial ones—into a model of a combination of the theories, say, their coproduct $T_1 + T_2$ (called *sum* by Hyland et al. [2006]).

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

(i) *The compositional approaches* combine models by their coproducts, which can be explicitly given in some cases, such as the coproduct of ideal monads [Ghani and Uustalu 2004].

(ii) *The incremental approaches*, such as monad transformers [Jaskelioff and Moggi 2010; Liang et al. 1995], layered monads [Filinski 1999], and modular handlers [Schrijvers et al. 2019a], turn existing models (in these cases, monads) into a new one, usually equipped with new operations.

The existing works (except Jaskelioff and Moggi [2010]) are not entirely satisfactory because most of them say little about how to *lift* operations on existing models to the combined (or transformed) models. Although there are canonical liftings for (first-order) *algebraic operations* on monoids M, which are operations of the form $A \square M \to M$ and commuting with monoid multiplication [Jaskelioff and Moggi 2010; Plotkin and Power 2003], there are many operations in practice outside this range:

(i) *Higher-order* operations, such as lambda abstraction [Fiore et al. 1999] and explicit substitution [Ghani et al. 2006], do not have the shape $A \square M \to M$.

(ii) *First-order non-algebraic* operations, called *scoped operations* by Piróg et al. [2018], have the shape $A \square M \to M$ but do not commute with monoid multiplication. Examples

include exception catching, acquiring a lock within a scope, parallel composition of processes.

Contributions. This paper develops a categorical framework of *modular models* of equational theories of monoids with a general form of operations, allowing modular models of different theories to be freely composable. Our development is parameterised by what we call a *monoidal theory family* $\mathcal{F}$, whose examples include the family of algebraic operations and the family of scoped operations. Formally, $\mathcal{F}$ is a category of equational theories of monoids with additional operations and equations in some monoidal category, such that the family is closed under coproducts and each theory admits free algebras. We represent equational theories using Fiore and Hur [2009]'s *equational systems* for their simplicity and generality, and we define a notion of *translations* between them to make them a category.

Each monoidal theory family $\mathcal{F}$ induces a category $\mathcal{F}$-**Mon** fibred over $\mathcal{F}$, whose objects are monoids equipped with a model of *some* theory in $\mathcal{F}$. Also, for each theory $\widehat{\Gamma} \in \mathcal{F}$, there is a fibred category $(\mathcal{F}+\widehat{\Gamma})$-**Mon** of monoids with both a $\widehat{\Gamma}$-model and a model of some theory in $\mathcal{F}$. A *modular model* of $\widehat{\Gamma}$ is then defined as a fibration morphism from $M : \mathcal{F}\text{-}\mathbf{Mon} \to (\mathcal{F}+\widehat{\Gamma})\text{-}\mathbf{Mon}$ together with a fibration 2-cell l. Intuitively, M can transform all models of all $\widehat{\Sigma} \in \mathcal{F}$ into models of $\widehat{\Sigma}+\widehat{\Gamma}$ in a uniform way, and l is a $\widehat{\Sigma}$-model homomorphism between the original and new models.

After introducing an *internal language* of monoidal categories (§2), which we will use to describe our theory, the structure and contributions of the paper are as follows:

- We recap *equational systems* and define *functorial translations* between them, making them a category, and we briefly discuss colimits in this category (§3).
- We define *monoidal theory families* and show that the family of algebraic operations is equivalent to the category of monoids, and it is a coreflective subcategory of all equational systems extending the theory of monoids (§4).
- We introduce *modular models* of a theory with some concrete examples. We also give general results for constructing modular models from monoid transformers (§5).
- We show how modular models are used for interpreting abstract syntax and a *fusion theorem* saying that sequentially applying two modular models to is equal to interpreting with the *composite* of two modular models (§6).

Finally we discuss related and future work (§7).

2 Monoids and Internal Languages

To put our work in context, we first review the concept of *monoids in monoidal categories* and some examples (§2.1). We then introduce an *internal language for monoidal categories*, which will be used in later sections (§2.2).

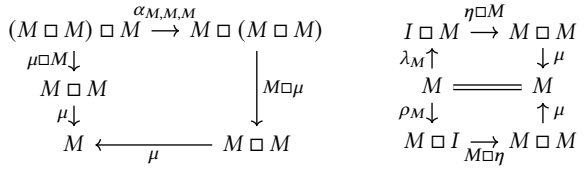


Figure 1. Commutative diagrams for the laws of monoids

2.1 Monoidal Categories and Monoids

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

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

satisfying some coherence axioms [Mac Lane 1998, §VII.1]. A monoidal category is *strict* if the isomorphisms are identity morphisms. 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* $\eta : I \to M$ out of the monoidal unit I of $\mathscr{E}$ making the diagrams in Figure 1 commute.

Example 2.1. 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 closed monoidal category. Particularly, the category **Set** of sets is a monoidal category in this way. Monoids in **Set** are precisely the usual notion of monoids in algebra.

Example 2.2. The category $\mathbf{Endo}(\mathscr{C})$ of endofunctors on some category $\mathscr{C}$ can be turned into a strict monoidal category with functor composition $F \circ G$ as the monoidal product and the identity functor $Id : \mathscr{C} \to \mathscr{C}$ as the unit.

Monoids in this category are precisely *monads* on $\mathscr{C}$, and they are used to model computational effects in programming languages [Moggi 1991], where the monoidal product $M \circ M \to M$ is understood as *sequential composition* of computations, and $Id \to M$ is understood as *pure computations*.

Example 2.3. The category $\mathbf{Endo}(\mathbf{Set})$ is not closed for size issues. But if we restrict it to the full subcategory $\mathbf{Endo}_f(\mathbf{Set})$ of *finitary endofunctors* on **Set**, it is indeed closed. More generally, $\mathbf{Endo}_f(\mathscr{C})$ is closed for all *locally finitely presentable* (lfp) categories $\mathscr{C}$ [Adamek and Rosicky 1994]. The right adjoint of $- \square G \dashv -/G$ is given by right Kan extension:

$$F/G = \int_{n \in \mathscr{C}_f} \prod_{\mathscr{C}(-,Gn)} Fn : \mathscr{C}_f \to \mathscr{C}$$

where $\mathscr{C}_f$ is the (essentially small) full subcategory of finitely presentable objects of $\mathscr{C}$.

Monoids in $\mathbf{Endo}_f(\mathscr{C})$ are called *finitary monads* and they are equivalent to *Lawvere $\mathscr{C}$-theories* [Nishizawa and Power

2009]. Computational effects modelled by finitary monads are usually called *algebraic effects* [Plotkin and Power 2004].

Example 2.4. For the category **Set**, or more generally any $\mathscr{C}$ that is lfp as a cartesian closed category [Kelly 1982], the category $\mathbf{Endo}_f(\mathscr{C})$ can be equipped with another monoidal structure. The monoidal unit is still the identity functor but the monoidal product is (a finitary version of) the Day convolution [Day 1970] induced by cartesian products:

$$F \star G = \int^{n,m \in \mathscr{C}_f \times \mathscr{C}_f} Fn \times Gm \times (-)^{n \times m}$$

We denote this monoidal category by $\mathbf{Endo}_f^{\star}(\mathscr{C})$. The product $\star$ is symmetric and closed with respect to

$$F/G = \int_{n \in \mathscr{C}_f} (G(- \times n))^{Fn}$$

Monoids in this monoidal category are called *applicative functors* or simply *applicatives* in the functional programming community, and they are used to model computational effects that do not allow effects to depend on previous effects [Mcbride and Paterson 2008; Paterson 2012].

Example 2.5. Let $\mathcal{F}$ be the category of finite sets and functions. The presheaf category $\mathbf{Set}^{\mathcal{F}}$ on $\mathcal{F}$ can be equipped with a (non-symmetric) monoidal structure

$$F \bullet G = \int^{n \in \mathcal{F}} Fn \times (G-)^n : \mathcal{F} \to \mathbf{Set}$$

with the inclusion functor $Vn = n$ as the unit.

Monoids in category $\mathbf{Set}^{\mathcal{F}}$ are used to model abstract syntax of variable-binding operations [Fiore and Hur 2010; Fiore et al. 1999]. Intuitively, objects in $\mathbf{Set}^{\mathcal{F}}$, i.e. functors $\mathcal{F} \to \mathbf{Set}$, represent indexed sets of terms indexed by the number of variables in context; the monoidal unit V is the (indexed) set of variables; the monoidal product $F \bullet G$ represents an explicit substitution [Ghani et al. 2006].

Remark. The reader might have noticed that $\mathbf{Set}^{\mathcal{F}}$ (Example 2.5) and $\mathbf{Endo}_f(\mathbf{Set})$ (Example 2.3) are equivalent monoidal categories. However, these two applications should be deemed as different. For example, when modelling abstract syntax, variable substitution $- \bullet G$ certainly commutes with lambda abstractions, whereas sequential composition $- \circ G$ does *not* commute with lambda abstractions viewed as a computational effect [van den Berg et al. 2021].

2.2 Internal Language for Monoidal Categories

When the monoidal category gets complex, commutative diagrams like those in Figure 1 can be unwieldy, so we use a small typed calculus introduced by Jaskelioff and Moggi [2010] as an *internal language* to denote constructions in monoidal categories. The use of the calculus greatly clarifies our presentation since it provides a notation for all monoidal categories as if we are working in the category of sets.

$$\frac{}{x : A \vdash x : A} \qquad \frac{f : A \to B \qquad \Gamma \vdash t : A}{\Gamma \vdash f(t) : B} \qquad \frac{}{\vdash * : \mathsf{I}}$$

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

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

Figure 2. Typing rules for the internal language

Syntax. The syntax of the calculus is as follows:

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

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

Typing. The type system in Figure 2 of the calculus is a substructural one, resembling Polakow and Pfenning [1999]'s *intuitionistic non-commutative linear logic*, since the language is to be interpreted in monoidal categories rather than only cartesian categories.

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

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

Additionally, given an interpretation $[\![f]\!] : [\![A]\!] \to [\![B]\!]$ in $\mathscr{E}$ for each primitive operation $f : A \to B$, all well typed terms $\Gamma \vdash t : A$ can be interpreted as morphisms $[\![t]\!] : [\![\Gamma]\!] \to [\![A]\!]$ in the evident way. For example, the typing rule for let is interpreted by arrow composition: [1]

$$[\![\mathsf{let} \ (x_1, x_2) = t_1 \ \mathsf{in} \ t_2 : B]\!] = [\![t_2]\!] \cdot ([\![\Gamma_l]\!] \sqcap [\![t_1]\!] \sqcap [\![\Gamma_r]\!])$$

Occasionally we want to be explicit about the denotations of base types and primitive operations that define $[\![-]\!]$. In this case we write them in the subscript like $[\![-]\!]_{\{\alpha \mapsto A\}}$.

Example 2.6. Assume a base type M and a primitive operation $\mu : M \sqcap M \to M$. The first diagram in the laws of monoids (Figure 1) can be denoted by the following pair of terms of the same type (throughout this paper, we write $\Gamma \vdash t_1 = t_2 : A$ for two terms t_1 and t_2 encoding an equation, but the equals sign here has no formal meaning):

$$\frac{\mu : M \sqcap M \to M}{x : M, y : M, z : M \vdash \mu(\mu(x, y), z) \ = \ \mu(x, \mu(y, z)) : M} \quad (1)$$

[1] We elide the use of associators α and unitors λ, ρ in the interpretation, since all the ways to apply them are coherent [Mac Lane 1998, §VII.2].

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

Extensions. Sometimes we work in monoidal categories with additional structure, and in this case we freely extend the calculus with new syntax for the additional structure. For example, when we work in closed monoidal categories, we extend the calculus with a new type constructor B/A, new syntax $\lambda x : A.\, t$ and $t_1\ t_2$ with typing rules

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

whose denotations are given by the corresponding structure of the closed monoidal category in the evident way:

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

where $abst : \mathscr{E}(C \square A, B) \to \mathscr{E}(C, B/A)$ is the natural isomorphism associated to the adjunction $(- \square A) \dashv (-/A)$ and $ev : (B/A) \square A \to B$ is its counit. Jaskelioff and Moggi [2010] use the notation B^A for this closed structure, while we use Lambek [1958]'s notation B/A to avoid the confusion with exponentials (right adjoint to cartesian products).

Example 2.7. Let $\mathscr{E}$ be a closed monoidal category. If some type M together with two terms

$$x : M, y : M \vdash \mu : M \qquad \text{and} \qquad \cdot \vdash \eta : M$$

denotes a monoid $\langle [\![M]\!], [\![\mu]\!], [\![\eta]\!] \rangle$ 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

$$f : M/M,\ g : M/M \vdash \lambda x.\, f\ (g\ x) : M/M \tag{2}$$

The embedding is given by $x : M \vdash \lambda y.\, m : M/M$.

This fact is frequently used in functional programming for optimising algorithms. When A is a free monoid in **Set**, this optimisation is known as *difference lists* [Hughes 1986]. When $\mathscr{E} = \mathbf{Endo}_f(\mathscr{C})$, this optimisation is known as *codensity transformation* [Hinze 2012].

3 Equational Systems and Translations

We have seen monoids in various monoidal categories, however in applications monoids are only interesting when they are equipped with additional operations and equational laws. For example, monads modelling effects all come with operations that model primitive effectful operations.

In this paper we use Fiore and Hur [2007, 2009]'s *equational systems* to deal with equational theories categorically and use their results to construct free algebras (abstract syntax) of theories, which we first recap below (§3.1). Then we extend their theory by introducing *translations* between equational systems, making them a category, and show some basic properties of colimits in this category (§3.2).

3.1 Equational Systems

An equational theory is characterised by the *signature* and *equations* of its operations. An abstract way to specify a signature is just using a functor $\Sigma : \mathscr{C} \to \mathscr{C}$, and then a Σ-*algebra* is a pair of a *carrier* $A \in \mathscr{C}$ and a *structure map* $\alpha : \Sigma A \to A$. For example, the theory of semigroups has exactly an associative binary operation, and thus signature functor $\Sigma_{\mathrm{SG}} = - \square -$, so a Σ_{SG}-algebra is equivalently an object A with an arrow $A \square A \to A$. We denote the category of Σ-algebras by Σ-**Alg**, whose arrows from $\langle A, \alpha \rangle$ to $\langle B, \beta \rangle$ are *algebra homomorphisms*, i.e. arrows $h : A \to B$ in $\mathscr{C}$ such that $h \cdot \alpha = \beta \cdot \Sigma h$. The forgetful functor dropping the structure map is denoted by $\mathbf{U}_\Sigma : \Sigma\text{-}\mathbf{Alg} \to \mathscr{C}$.

Equations of theories are usually informally presented as commutative diagrams containing a formal arrow $\Sigma A \to A$ (like the first diagram in Figure 1 for associativity of a binary operation). One way to formulate such a diagram is a pair of functors $L, R : \Sigma\text{-}\mathbf{Alg} \to \Gamma\text{-}\mathbf{Alg}$ where the functor Γ encodes the starting node of the diagram, and L and R represent the two paths of the diagram.

Definition 3.1 (Fiore and Hur [2009])**.** An *equational system* $\widehat{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ 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 *model* of $\widehat{\Sigma}$ 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 $\widehat{\Sigma}$-algebras is denoted by $\widehat{\Sigma}$-**Alg**.

Among the algebras of an equational theory $\widehat{\Sigma}$ over $\mathscr{C}$, the *free algebras* are particularly important in computing science since they model abstract syntax of terms of the theory. Fiore and Hur [2009] show various conditions for the existence of free algebras, i.e. the left adjoint to the forgetful functor $\widehat{\Sigma}\text{-}\mathbf{Alg} \to \mathscr{C}$. In this paper, we will use the following one.

Proposition 3.2 (Fiore and Hur [2009])**.** *For all equational systems $\widehat{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ over $\mathscr{C}$, if $\mathscr{C}$ is cocomplete and Σ and Γ preserve colimits of ω-chains, then there are left adjoints*

$$\widehat{\Sigma}\text{-}\mathbf{Alg} \underset{\longrightarrow}{\overset{\longleftarrow}{\bot}} \Sigma\text{-}\mathbf{Alg} \underset{\longrightarrow}{\overset{\longleftarrow}{\bot}} \mathscr{C}$$

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

Compared to alternatives such as enriched algebraic theories [Kelly and Power 1993] and Lawvere theories [Power 1999], equational systems are both simpler and more general, yet still offer strong results for the existence of free algebras. Furthermore, we use the internal language in §2.2 to specify the data for equational systems (on monoidal categories) in a syntactic way (in fact, we already did this in Example 2.6):

(i) To give a *functorial signature/context* $\mathscr{E} \to \mathscr{E}$, we adjoin a distinguished base type τ to the internal language, and

then every type expression T_τ in which τ occurs positively[2] induces a functor $A \mapsto [\![T_\tau]\!]_{\{\tau \mapsto A\}}$. The arrow mapping for the functor is the evident one following the functoriality of the type constructors, and we also add a new term syntax $T_\tau f$ with the following typing rule for the arrow mapping:

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

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

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

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

By structural induction on typing derivations in the style of Reynolds [1983]'s abstraction theorem, it is possible to show that $\mathbf{U}_\Gamma \circ F = \mathbf{U}_\Sigma$ required in Definition 3.1. For example, the two terms (1) denote a pair of functorial terms

$$(- \square -)\text{-}\mathbf{Alg} \to (- \square - \square -)\text{-}\mathbf{Alg}$$

Monoids. The notion of monoids (§2.1) can be reformulated as an equational system:

$$\mathbf{Mon} = (\Sigma_{\mathbf{Mon}} \triangleright \Gamma_{\mathbf{Mon}} \vdash L_{\mathbf{Mon}} = R_{\mathbf{Mon}}) \tag{3}$$

Assuming that $\mathscr{E}$ has coproducts, the internal language can be extended with a type constructor $A_1 + \cdots + A_n$ with the customary syntax for elimination $[t_1, \ldots, t_n]$ and introduction $\mathrm{inj}_i\ t$. Then the functorial signature and context are

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

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

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

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

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

As demonstrated above, although equational systems as in Definition 3.1 only have one functorial signature Σ and equation $\Gamma \vdash L = R$, multiple operations and equations can be expressed using coproducts. Thus when the monoidal category has coproducts, we will just informally say an equational system with a set of operations and a set E of equations $(\Gamma_i \vdash L_i = R_i)_{i \in E}$. Also, we denote an equational system $\widehat{\Sigma}$ extended with an equation by $\widehat{\Sigma} \uplus (\Gamma \vdash L = R)$.

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

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

for all X in $\mathscr{E}$ and $\langle Y \in \mathscr{E}, f : I \to \mathscr{E} \rangle$ in the coslice category $I/\mathscr{E}$, satisfying coherence conditions analogous to those of strengths [Fiore and Hur 2009, §7.2.1]. To denote Σ and θ syntactically, we extend the internal language with a type constructor Σ and the following typing rules

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

Then the theory Σ-**Mon** of *Σ-monoids*

$$\Sigma\text{-}\mathbf{Mon} = (\Sigma + \Sigma_{\mathbf{Mon}} \triangleright \Gamma_{\mathbf{Mon}} \vdash L_{\mathbf{Mon}} = R_{\mathbf{Mon}}) \tag{4}$$
$$\uplus\ ((\Sigma -) \square - \vdash L_{\Sigma\text{-}\mathbf{Mon}} = R_{\Sigma\text{-}\mathbf{Mon}})$$

extends the theory **Mon** of monoids (3) with a new operation $\mathrm{op} : \Sigma\tau \to \tau$ and a new equation $L_{\Sigma\text{-}\mathbf{Mon}} = R_{\Sigma\text{-}\mathbf{Mon}}$ given by

$$x : \Sigma\tau, y : \tau \vdash \mu(\mathrm{op}\ x, y) = \mathrm{op}((\Sigma\mu)(\theta_{\tau,\langle\tau,\eta\rangle}(x, y))) : \tau \tag{5}$$

which encodes the following commutative diagram:

$$\begin{array}{ccccc} (\Sigma\tau) \square \tau & \xrightarrow{\theta_{\tau,\langle\tau,\eta\rangle}} & \Sigma(\tau \square \tau) & \xrightarrow{\Sigma\mu} & \Sigma\tau \\ {\scriptstyle \mathrm{op}\square\tau}\downarrow & & & & \downarrow{\scriptstyle \mathrm{op}} \\ \tau \square \tau & & \xrightarrow[\mu]{} & & \tau \end{array}$$

The special case of (5) for $\Sigma = A \square -$ (with the identity pointed strength $\theta_{X,\langle Y,f\rangle} = id$) is exactly *algebraicity* of operations op on monoids τ [Jaskelioff and Moggi 2010], and such an operation op is called an *algebraic operation*. Thus we call (5) *generalised algebraicity*.

Now let us look at some concrete examples. In all the following examples, category $\mathscr{E}$ is assumed to have set-indexed coproducts $\coprod_{i \in S} A_i$ and finite products $\prod_{i \in F} A_i$.

Example 3.3. Letting E be a set, the theory $\mathbf{Exc}_E$ of *exceptions* is an instance of Σ-monoids for which $\Sigma = (\coprod_E 1) \square -$ where $\coprod_E 1$ is the E-fold coproduct of the terminal object in $\mathscr{E}$ (which may be different from the monoidal unit I). For $\mathscr{E} = \mathbf{Endo}_f(\mathscr{C})$ (Example 2.3), the theory $\mathbf{Exc}_E$ describes monads M equipped with a natural transformation

$$(\coprod_E 1) \circ M = \coprod_E (1 \circ M) = \coprod_E 1 \to M : \mathbf{Endo}_f(\mathscr{C})$$

whose component $1 \to M$ for each $e \in E$ represents a computation throwing an exception e [Plotkin and Power 2002].

Working in monoidal categories in the abstract allows us to interpret the theory $\mathbf{Exc}_E$ of exceptions in other settings: taking $\mathscr{E} = \mathbf{Endo}^\star_f(\mathscr{C})$ (Example 2.4), the theory describes applicative functors with exception throwing; and taking $\mathscr{E} = \mathbf{Set}^{\mathcal{F}}$ (Example 2.5), the theory describes an abstract syntax with E nullary term constructors.

[2] The rule for positivity is standard: τ occurs in itself positively and all other base types β both positively and negatively; and τ occurs in B/A positively (negatively) if τ occurs in B positively (negatively) and in A negatively (positively), and so on for other type constructors.

Example 3.4. Although exception *throwing* is an algebraic operation in the sense of Plotkin and Power [2003], it is well known that exception *catching* is not, as it does not commute with sequential composition of computations:

$$\frac{\mathit{catch} : \tau \times \tau \to \tau}{p : \tau, h : \tau, k : \tau \vdash (\mathit{catch}\ \langle p, h\rangle;\ k) \neq \mathit{catch}\ \langle (p; k), (h; k)\rangle : \tau}$$

where $(p; q)$ denotes multiplication $\mu(p, q)$ of monads , and $\langle x, y\rangle$ is the term syntax for introducing a cartesian product $X \times Y$. Thus *catch* cannot be modelled with as Σ-monoids with an operation $\tau \times \tau \to \tau$, or it will contradict (5).

A solution advocated by Piróg et al. [2018] is to model exception catching as a *scoped operation*. In terms of Σ-monoids, their solution models exception throwing and catching as Σ_{TC}-monoids in $\mathbf{Endo}_f(\mathscr{C})$ where

$$\Sigma_{\mathrm{TC}} = (1 \circ -) + (I \times I) \circ - \circ - : \mathbf{Endo}_f(\mathscr{C}) \to \mathbf{Endo}_f(\mathscr{C})$$

with the following pointed strength $\theta_{X,\langle Y,f\rangle}$ for all $X, Y \in \mathbf{Endo}_f(\mathscr{C})$ and $f : I \to Y$:

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

where the boxed Y is inserted using $f : I \to Y$. The intuition for the signature Σ_{TC} is that $1 \circ -$ stands for *throw*ing an exception like in Example 3.3, and the second component

$$(I \times I) \circ - \circ - \quad \cong \quad (- \times -) \circ -$$

stands for catching—a binary operation *catch* that also takes an *explicit continuation* [Piróg et al. 2018]. For Σ_{TC} and the pointed strength θ above, the generalised algebraicity (5) means the following equation for *catch*:

$$\mathit{catch}(\langle p, h\rangle, t); k = \mathit{catch}(\langle p, h\rangle, (t; k)) \tag{6}$$

which is semantically correct for exception catching.

Additionally, we can characterise the interaction of exception and catching by the following equations:

$$\begin{aligned} k : \tau &\vdash \mathit{catch}(\langle \mathit{throw}, \eta\rangle, k) = k : \tau \\ k : \tau &\vdash \mathit{catch}(\langle \mathit{throw}, \mathit{throw}\rangle, k) = \mathit{throw} : \tau \\ k : \tau &\vdash \mathit{catch}(\langle \eta, \mathit{throw}\rangle, k) = k : \tau \\ k : \tau &\vdash \mathit{catch}(\langle \eta, \eta\rangle, k) = k : \tau \end{aligned} \tag{7}$$

where $\eta : \mathsf{I} \to \tau$, $\mathit{throw} : 1 \to \tau$, and $\mathit{catch} : (\tau \times \tau) \square \tau \to \tau$. Also, these equations can be alternatively presented with an empty context by replacing all the k's with η like

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

which is equivalent to (7), since by the generalised algebraicity (6), $\mathit{catch}(\langle x, y\rangle, \eta); k = \mathit{catch}(\langle x, y\rangle, k)$.

Example 3.5. Letting S be a finite set, the theory $\mathbf{St}_S$ of *monads with global S-state* [Plotkin and Power 2002] can be generally defined for monoids as follows. The theory $\mathbf{St}_S$ is $\Sigma_{\mathbf{St}_S}$-**Mon** with signature $\Sigma_{\mathbf{St}_S} = ((\prod_S \mathsf{I}) \square \tau) + ((\coprod_S \mathsf{I}) \square \tau)$, whose first part 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

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

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].

3.2 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 naturality between functors on equational systems. A natural idea for arrows from an equational theory $\widehat{\Sigma}$ to $\widehat{\Gamma}$ is a *translation* from operations in $\widehat{\Sigma}$ to terms of $\widehat{\Gamma}$, and Fiore and Mahmoud [2010] show that using translations as arrows allows equivalences between equational theories and Lawvere theories to be made. In the following, we tailor translations to the setting of equational systems.

Definition 3.6. A *functorial translation* of equational systems from $\widehat{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ to $\widehat{\Sigma}' = (\Sigma' \triangleright \Gamma' \vdash L' = R')$ is a functor $T : \widehat{\Sigma}'\text{-}\mathbf{Alg} \to \widehat{\Sigma}\text{-}\mathbf{Alg}$ that is identity on carriers and homomorphisms. Equational systems on $\mathscr{C}$ and their translations form a category $\mathbf{Eqs}(\mathscr{C})$, in which the identity arrows are the identity functors $\widehat{\Sigma}\text{-}\mathbf{Alg} \to \widehat{\Sigma}\text{-}\mathbf{Alg}$, and composition of translations $T \circ T'$ are functor composition.

Example 3.7. Consider two instances $\mathbf{St}_S$ and $\mathbf{St}_{S'}$ of the theory in Example 3.5 with a bijection $f : S \cong S'$. An example of a translation $\mathbf{St}_S \to \mathbf{St}_{S'}$ is $T : \mathbf{St}_{S'}\text{-}\mathbf{Alg} \to \mathbf{St}_S\text{-}\mathbf{Alg}$ mapping $\langle A, \alpha : (\Sigma_{\mathbf{St}_{S'}} + \Sigma_{\mathbf{Mon}})A \to A\rangle$ to A with

$$[\alpha \cdot \iota_1 \cdot \tilde{f},\ \ \alpha \cdot \iota_2] : (\Sigma_{\mathbf{St}_S} + \Sigma_{\mathbf{Mon}})A \to A$$

where $\tilde{f} : \Sigma_{\mathbf{St}_S} \to \Sigma_{\mathbf{St}_{S'}}$ is the evident natural isomorphism from f. This translation is an isomorphism in $\mathbf{Eqs}(\mathscr{E})$.

Colimits in $\mathbf{Eqs}(\mathscr{C})$ allows one to combine equational systems. The following are some elementary results about their existence, whose proofs can be found in Appendix A.

Lemma 3.8. *(i) When $\mathscr{C}$ has all (small) coproducts, so does the category* $\mathbf{Eqs}(\mathscr{C})$. *(ii) When $\mathscr{C}$ has binary coproducts and there is a reflection $\Sigma\text{-}\mathbf{Alg} \rightleftarrows \widehat{\Sigma}\text{-}\mathbf{Alg}$, then all pairs of translations $T_1, T_2 : \widehat{\Gamma} \to \widehat{\Sigma}$ has coequalisers.*

Theorem 1. *The full subcategory $\mathbf{Eqs}_\omega(\mathscr{C}) \subseteq \mathbf{Eqs}(\mathscr{C})$ of equational systems whose functorial signature and context preserve colimits of ω-chains is cocomplete if $\mathscr{C}$ is cocomplete.*

4 Monoidal Theory Families

In applications, we are usually interested in a family of theories of Σ-monoids for $\Sigma : \mathscr{E} \to \mathscr{E}$ of a certain form, such as Σ of the form $A \square -$ for algebraic operations, and Σ of the form $A \square - \square -$ for scoped operations. In this section, we capture them as *monoidal theory families* and study some prominent examples and their connections.

Definition 4.1. A *monoidal theory family* over a monoidal category $\mathscr{E}$ is a full subcategory $\mathcal{F} \subseteq \mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$ of the coslice category of equational systems under the theory $\mathbf{Mon}$ of monoids such that

(i) $\mathcal{F}$ is closed under coproducts in $\mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$, and
(ii) each $\langle \widehat{\Sigma}, T\rangle \in \mathcal{F}$ admits free algebras $\mathscr{E} \to \widehat{\Sigma}\text{-}\mathbf{Alg}$.

Remark. Note that the coproducts in $\mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$ are equivalently pushouts $\widehat{\Sigma} \leftarrow \mathbf{Mon} \to \widehat{\Gamma}$ in $\mathbf{Eqs}(\mathscr{E})$, so coproducts in $\mathcal{F}$ are intuitively combining theories while identifying their monoid operations.

Algebraic Operations. Our first example is the family $\mathbf{ALG}$ of *algebraic operations*. For $\mathscr{E}$ cocomplete with $\square : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ preserving coproducts in the first argument and colimits of ω-chains in both arguments. The full subcategory $\mathbf{ALG}(\mathscr{E})$ of $\mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$ contains equational systems

$$\Sigma\text{-}\mathbf{Mon} \uplus Eq, \text{ for all } \Sigma \in \{A \square - \mid A \in \mathscr{E}\} \tag{8}$$

and for all functorial equations $Eq = (E \vdash L = R)$ where E is a *constant functor* (see the remark below). In other words, $\mathbf{ALG}(\mathscr{E})$ contains all theories extending some theory Σ-$\mathbf{Mon}$ (4) (for $\Sigma = A \square -$ and the identity pointed strength) with an equation. Each equation system (8) is equipped with the inclusion translation to form subcategory of $\mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$:

$$\mathbf{Mon} \hookrightarrow \Sigma\text{-}\mathbf{Mon} \hookrightarrow \big(\Sigma\text{-}\mathbf{Mon} \uplus Eq\big)$$

The category $\mathbf{ALG}(\mathscr{E})$ satisfies the conditions in Definition 4.1 since it has coproducts by taking the coproducts of signatures and equations respectively:

$$\begin{aligned}&\textstyle\coprod_{i\in I} \big((A_i \square -)\text{-}\mathbf{Mon} \uplus (E_i \vdash L_i = R_i)\big)\\ &= \big(\textstyle\coprod_i A_i \square -\big)\text{-}\mathbf{Mon} \uplus \big(\textstyle\coprod_i E_i \vdash [L_i]_{i\in I} = [R_i]_{i\in I}\big)\end{aligned}$$

and all equational systems (8) in $\mathbf{ALG}(\mathscr{E})$ have free algebras by Proposition 3.2, since the signature and context functor of Σ-$\mathbf{Mon}$ preserve colimits of ω-chains.

Remark. The restriction in (8) that the functorial context must be a constant functor E needs some explanation: for a functor $\Sigma : \mathscr{E} \to \mathscr{E}$ with initial algebra Σ^* and a constant functor E mapping to an object $E \in \mathscr{E}$, it is possible to show that arrows $E \to \Sigma^*$ in $\mathscr{E}$ are equivalently functorial terms $\Sigma\text{-}\mathbf{Alg} \to E\text{-}\mathbf{Alg}$. Thus Eq in (8) is equivalently a pair of arrows $E \to (\Sigma + \Sigma_{\mathbf{Mon}})^*$, which can be understood as $|E|$-pairs of terms built from the operations, and indeed this is Kelly and Power [1993]'s way to specify equations.

For $\mathscr{E} = \mathbf{Endo}_f(\mathscr{C})$ over an lfp category $\mathscr{C}$, the family $\mathbf{ALG}(\mathscr{E})$ consists of theories of finitary monads over $\mathscr{C}$ with algebraic operations, and in particular the theory of exceptions (Example 3.3) is an object of $\mathbf{ALG}(\mathscr{E})$. More generally, we have the following characterisation of $\mathbf{ALG}(\mathscr{E})$.

Theorem 2. *For a monoidal cocomplete category $\mathscr{E}$ with $\square : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ preserving coproducts in the first argument and colimits of ω-chains in both arguments, there is an equivalence $\mathbf{Mon}(\mathscr{E}) \cong \mathbf{ALG}(\mathscr{E})$ of categories where $\mathbf{Mon}(\mathscr{E})$ is the category of monoids in $\mathscr{E}$.*

In particular, when $\mathscr{E} = \mathbf{Endo}_f(\mathscr{C})$, $\mathbf{Mon}(\mathscr{E})$ is exactly the category of finitary monads, and $\mathbf{ALG}(\mathbf{Endo}_f(\mathscr{C}))$ is a (slightly unconventional) way to formulate first-order equational theories and translations. Thus this theorem can be seen as a generalisation of the classical equivalence between finitary monads and (first-order) equational theories [Linton 1966; Mahmoud 2010] to the setting of monoids.

Another important property of $\mathbf{ALG}(\mathscr{E})$ is that all equational theories of monoids with additional operations can be cast into one in $\mathbf{ALG}(\mathscr{E})$ by a coreflection.

Theorem 3. *Let $\mathscr{E}$ be a monoidal cocomplete category with $\square : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ preserving coproducts in the first argument and colimits of ω-chains in both arguments, $\mathbf{ALG}(\mathscr{E})$ is a coreflective subcategory of $\mathbf{Mon}/\mathbf{Eqs}_\omega(\mathscr{E})$:*

$$\mathbf{Mon}(\mathscr{E}) \underset{\cong}{\overset{\longleftarrow}{\longrightarrow}} \mathbf{ALG}(\mathscr{E}) \underset{\top}{\overset{\longleftarrow}{\hookrightarrow}} \mathbf{Mon}/\mathbf{Eqs}_\omega(\mathscr{E})$$

Scoped Operations. Another monoidal theory family relevant to computational effects is the family $\mathbf{SCP}(\mathscr{E})$ of *scoped (and algebraic) operations*, such as exception catching (Example 3.4). The family $\mathbf{SCP}(\mathscr{E})$ is given by the full subcategory of $\mathbf{Mon}/\mathbf{Eqs}(\mathscr{E})$ containing equational systems

$$\Sigma\text{-}\mathbf{Mon} \uplus Eq, \text{ for all } \Sigma \in \{(A\square - \square -) + (B \square -) \mid A, B \in \mathscr{E}\} \tag{9}$$

and for all functorial equations $Eq = (E \vdash L = R)$ with a constant functorial context E. The pointed strength of Σ needed in the definition of Σ-$\mathbf{Mon}$ (4) is

$$\begin{aligned}(\Sigma X) \square Y \cong A \square X \square X \square Y + B \square X \square Y &\xrightarrow{A\square X \square \eta^Y + id}\\ A \square X \square Y \square X \square Y + B \square X \square Y &\cong \Sigma(X \square Y)\end{aligned}$$

Each equational system (9) is equipped with the inclusion translation $\mathbf{Mon} \to \Sigma\text{-}\mathbf{Mon} \uplus (E \vdash L = R)$.

Piróg et al. [2018] introduced scoped operations to model non-algebraic effects that delimit scopes, such as catching exceptions and parallel composition of processes. Indeed, the signature Σ (9) has two parts: $B \square -$ represents a usual algebraic operation as in $\mathbf{ALG}$, whereas $A \square - \square -$ represents an operation that is not necessarily algebraic, since arrows $f : A \square M \square M \to M$ on a monoid M satisfying generalised algebraicity (5) are equivalently arrows $g : A \square M \to M$

without algebraicity, with the mappings given by

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

Yang et al. [2022] show that scoped operations can be encoded by algebraic operations in $\mathbf{Endo}_f(\mathscr{C})$, which is true more generally following Theorem 3.

Proposition 4.2. *The family* $\mathbf{ALG}(\mathscr{E})$ *of algebraic operations is a coreflective subcategory of* $\mathbf{SCP}(\mathscr{E})$*, thus:*

$$\mathscr{E} \underset{\longleftarrow}{\overset{\longrightarrow}{\bot}} \mathbf{Mon}(\mathscr{E}) \underset{\longleftarrow}{\overset{\longrightarrow}{\cong}} \mathbf{ALG}(\mathscr{E}) \underset{\longleftarrow}{\overset{\longrightarrow}{\bot}} \mathbf{SCP}(\mathscr{E})$$

Variable-Binding Operations. Our final example is the monoidal theory family of *variable-binding operations* [Fiore et al. 1999]. For this example, we work concretely in the monoidal category $\mathbf{Set}^{\mathcal{F}}$ (Example 2.5) for simplicity, although it is possible to generalise to monoidal categories with axiomatic structure like in the previous examples.

A *binding signature* $\langle O, a\rangle$ consists of a set O of operations and an assignment $a : O \to \mathbb{N}^*$ of arity to each operation. If $a(o) = \langle n_i\rangle_{1\leqslant i\leqslant k}$ for some $o \in O$, then o stands for an operation taking k arguments, each binding n_i variables:

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

A binding signature determines a functor $\Sigma : \mathbf{Set}^{\mathcal{F}} \to \mathbf{Set}^{\mathcal{F}}$

$$\Sigma = \coprod_{o\in O} \prod_{1\leqslant i\leqslant k} (-)^{V^{n_i}} \tag{10}$$

where $a(o) = \langle n_i\rangle_{1\leqslant i\leqslant k}$, and $(-)^{V^{n_i}}$ is the exponential by n_i-fold product of the monoidal unit $V = - : \mathcal{F} \to \mathbf{Set}$. Equipped with a suitable pointed strength [Fiore et al. 1999], each Σ defines a theory Σ-**Mon** of Σ-monoids (4). The monoidal theory family $\mathbf{VAR}(\mathbf{Set}^{\mathcal{F}})$ then contains all theories (equipped with inclusion translations from **Mon**)

$$\Sigma\text{-}\mathbf{Mon} \uplus (\Gamma \vdash L = R), \text{ for all } \Sigma \text{ in the form of (10)}$$

and all equations $\Gamma \vdash L = R$ with $\Gamma : \mathbf{Set}^{\mathcal{F}} \to \mathbf{Set}^{\mathcal{F}}$ also taking the form of (10).

Proposition 4.3. *The family* $\mathbf{ALG}(\mathbf{Set}^{\mathcal{F}})$ *of algebraic operations is a coreflective subcategory of* $\mathbf{VAR}(\mathbf{Set}^{\mathcal{F}})$.

5 Modular Models of Monoids

In this section we present our main definition, a formulation of *modular models* of theories in a monoidal theory family $\mathcal{F}$ such that modular models of each theory $\widehat{\Sigma} \in \mathcal{F}$ are composable with models of another theory $\widehat{\Gamma} \in \mathcal{F}$, giving rise to models of the combined theory $\widehat{\Sigma} + \widehat{\Gamma}$. Our formulation is organised by the means of *fibrations over* $\mathcal{F}$ (§5.1). After spelling out the definition of modular models, we show concrete examples of modular models for the theory of exception throwing (Example 5.7) and catching (Example 5.8), together with theorems constructing modular models for algebraic operations (Theorem 4) and scoped operations (Theorem 5) from *monoid transformers* [Jaskelioff and Moggi 2010].

5.1 Fibrations and $\mathcal{F}$-Monoids

Fixing a monoidal theory family $\mathcal{F}$ and some theory $\widehat{\Sigma} \in \mathcal{F}$, we would like modular models of $\widehat{\Sigma}$ to be able to transform all models of all theories $\widehat{\Gamma} \in \mathcal{F}$ into models of $\widehat{\Sigma} + \widehat{\Gamma}$ in a uniform way, so we need a way to talk about *all models of all theories* in $\mathcal{F}$ conveniently. For this purpose, we use *(split) fibrations over* $\mathcal{F}$, which we briefly recap below and refer the reader to Jacobs [1999] for a proper introduction.

Definition 5.1 (Split Fibrations)**.** Let $P : \mathscr{T} \to \mathscr{B}$ be any functor. An arrow $f : X \to Y$ in $\mathscr{T}$ is said to be *cartesian* if for every $g : Z \to Y$ such that $Pg = Pf \cdot w$ with some $w : PZ \to PX$, there is a unique $h : Z \to X$ satisfying $Ph = w$ and $f \cdot h = g$:

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

A *split fibration over* $\mathscr{B}$ is a functor $P : \mathscr{T} \to \mathscr{B}$ equipped with a mapping κ, called a *split cleavage*, sending every pair of an arrow $u : I \to J \in \mathscr{B}$ and an object $Y \in \mathscr{T}$ with $PY = J$ to a cartesian morphism $\kappa(Y, u) : X \to Y$ in $\mathscr{T}$ such that $P\kappa(Y, u) = u$. The mapping κ is required to preserve identities and composition:

$$\kappa(Y, id_J) = id_Y \qquad \kappa(Y, u \cdot v) = \kappa(Y, u) \cdot \kappa(X, v)$$

where X is the domain of $\kappa(Y, u)$. The category $\mathscr{T}$ is called the *total category* and $\mathscr{B}$ is called the *base category*.

Split fibrations form a category $\mathbf{Fib}_s$ in which an arrow between split fibrations $\langle P, \kappa\rangle$ and $\langle P', \kappa'\rangle$ is a pair of functors $\langle F, G\rangle$ making the left diagram in (11) commute and preserving the cleavage $F\kappa(Y, u) = \kappa(FY, Gu)$.

$$\begin{array}{ccc} \mathscr{T} & \xrightarrow{F} & \mathscr{T}' \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P'} \\ \mathscr{B} & \xrightarrow[G]{} & \mathscr{B}' \end{array} \qquad \begin{array}{ccc} \mathscr{T} & \overset{F}{\underset{F'}{\Downarrow\sigma}} & \mathscr{T}' \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P'} \\ \mathscr{B} & \underset{G}{\overset{G'}{\Uparrow\tau}} & \mathscr{B}' \end{array} \tag{11}$$

The identity arrows and composition are respectively pairs of identity functors and functor composition. The category $\mathbf{Fib}_s$ can also be extended to a 2-category, for which a 2-cell is also a pair $\langle \sigma : F \to F', \tau : G \to G'\rangle$ such that $P' \circ \sigma = \tau \circ P$ as in the right diagram in (11).

$\mathcal{F}$-Monoids. Given a monoidal theory family $\mathcal{F}$ over some $\mathscr{E}$, we would like to construct a category $\mathcal{F}$-**Mon** of all models of all theories in $\mathcal{F}$ and a split fibration $P : \mathcal{F}\text{-}\mathbf{Mon} \to \mathcal{F}$. These can be constructed using the *Grothendieck construction* for the functor $(-)\text{-}\mathbf{Alg} : \mathcal{F} \to \mathbf{Eqs}(\mathscr{E}) \to \mathbf{CAT}$, or explicitly given by the following definition.

Definition 5.2. The objects of category $\mathcal{F}$-**Mon** are tuples

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

such that $\langle \widehat{\Sigma}, T_\Sigma\rangle \in \mathcal{F}$ and $\langle A, \alpha\rangle \in \widehat{\Sigma}\text{-}\mathbf{Alg}$. Arrows between two objects $\langle \widehat{\Sigma}, T_\Sigma, A, \alpha\rangle$ and $\langle \widehat{\Gamma}, T_\Gamma, B, \beta\rangle$ are pairs $\langle T, h\rangle$ where

$T : \widehat{\Sigma} \to \widehat{\Gamma}$ is a functorial translation in $\mathcal{F}$, i.e. functors $\widehat{\Gamma}$-**Alg** $\to \widehat{\Sigma}$-**Alg** that are identity on carriers and homomorphisms and $T \circ T_\Sigma = T_\Gamma$; the other component $h : A \to B \in \mathscr{E}$ is a $\widehat{\Sigma}$-algebra homomorphism from $\langle A, \alpha\rangle$ to $T\langle B, \beta\rangle$:

$$\begin{array}{ccc} \Sigma A & \xrightarrow{\alpha} & A \\ {\scriptstyle \Sigma h}\downarrow & & \downarrow{\scriptstyle h} \\ \Sigma B & \xrightarrow[T\langle B,\beta\rangle]{} & B \end{array} \qquad \overset{T}{\longleftarrow} \qquad \begin{array}{ccc} \Gamma B & \xrightarrow{\beta} & B \end{array}$$

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

The split fibration $P : \mathcal{F}$-**Mon** $\to \mathcal{F}$ is the projection:

$$P\langle \widehat{\Sigma}, T_\Sigma, A, \alpha\rangle = \langle \widehat{\Sigma}, T_\Sigma\rangle \quad \text{and} \quad P\langle T, h\rangle = T$$

and it is equipped with a split cleavage sending arrows $T : \langle \widehat{\Sigma}, T_\Sigma\rangle \to \langle \widehat{\Gamma}, T_\Gamma\rangle \in \mathcal{F}$ and objects $\langle \widehat{\Gamma}, T_\Gamma, B, \beta\rangle \in \mathcal{F}$-**Mon** to

$$\langle T, id\rangle : \langle \widehat{\Sigma}, T_\Sigma, B, T\langle B, \beta\rangle\rangle \to \langle \widehat{\Gamma}, T_\Gamma, B, \beta\rangle \tag{12}$$

Example 5.3. Let $\mathcal{F}$ be **ALG**$(\mathscr{E})$ and $\mathscr{E}$ be **Endo**$_f$(**Set**), the category of finitary endofunctors on **Set**. Then the objects of $\mathcal{F}$-**Mon** are all finitary monads $\langle M, \eta^M, \mu^M\rangle$ on **Set**, each equipped with an algebraic operation $A \circ M \to M$ satisfying some equation $E \vdash L = R$. An arrow between two such objects $\langle M, \eta^M, \mu^M, \alpha\rangle$ and $\langle N, \eta^N, \mu^N, \beta\rangle$ is a pair $\langle T, h\rangle$, where T is a translation between the associated algebraic theories, and $h : M \to N \in$ **Endo**$_f$(**Set**) is both a monad morphism and an algebra homomorphism from M to $T\langle N, \eta^N, \mu^N, \beta\rangle$.

Given a theory $\langle \widehat{\Gamma}, T_\Gamma\rangle \in \mathcal{F}$, we are also interested in all models of all theories in $\mathcal{F}$ that are additionally equipped with a $\widehat{\Gamma}$-algebra. Such $(\mathcal{F}+\widehat{\Gamma})$-*monoids* can be obtained easily by a *change-of-base* for the fibration $P : \mathcal{F}$-**Mon** $\to \mathcal{F}$ along the functor $(- + \widehat{\Gamma}) : \mathcal{F} \to \mathcal{F}$, which is just the following pullback in the category of categories:

$$\begin{array}{ccc} (\mathcal{F}+\widehat{\Gamma})\text{-}\mathbf{Mon} & \xrightarrow{K} & \mathcal{F}\text{-}\mathbf{Mon} \\ {\scriptstyle Q}\downarrow & \lrcorner & \downarrow{\scriptstyle P} \\ \mathcal{F} & \xrightarrow[-+\widehat{\Gamma}]{} & \mathcal{F} \end{array} \tag{13}$$

Explicitly, the objects of $(\mathcal{F} + \widehat{\Gamma})$-**Mon** are tuples

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

such that $\langle \widehat{\Sigma}, T_\Sigma\rangle \in \mathcal{F}$, $\langle A, \alpha\rangle \in \widehat{\Sigma}$-**Alg**, $\langle A, \beta\rangle \in \widehat{\Gamma}$-**Alg**, and

$$T_\Gamma\langle A, \alpha\rangle = T_\Sigma\langle A, \beta\rangle \in \mathbf{Mon}(\mathscr{E})$$

Arrows in $(\mathcal{F} + \widehat{\Gamma})$-**Mon** are the same as those $\langle T, h\rangle$ in $\mathcal{F}$-**Mon** (12), but require h also to be a $\widehat{\Gamma}$-homomorphism.

The pullback (13) also induces two functors Q and K. The functor $Q : (\mathcal{F} + \widehat{\Gamma})$-**Mon** $\to \mathcal{F}$ is the projection to $\mathcal{F}$, and it is a split fibration with a split cleavage similar to that of P (12), sending pairs of objects $\langle \widehat{\Sigma}, T_\Sigma, A, \alpha, \beta\rangle$ in $(\mathcal{F} + \widehat{\Gamma})$-**Mon** and translations $T : \langle \widehat{\Phi}, T_\Phi\rangle \to \langle \widehat{\Sigma}, T_\Sigma\rangle$ in $\mathcal{F}$ to

$$\langle T, id\rangle : \langle \widehat{\Phi}, T_\Phi, A, T\langle A, \alpha\rangle, \beta\rangle \to \langle \widehat{\Sigma}, T_\Sigma, A, \alpha, \beta\rangle$$

The functor $K : (\mathcal{F} + \widehat{\Gamma})$-**Mon** $\to \mathcal{F}$-**Mon** maps objects $\langle \widehat{\Sigma}, T_\Sigma, A, \alpha, \beta\rangle$ to $\langle\langle \widehat{\Sigma}, T_\Sigma\rangle + \langle \widehat{\Gamma}, T_\Gamma\rangle, A, [\alpha, \beta]\rangle$. Additionally, the pair $\langle K, - + \widehat{\Gamma}\rangle$ is a split fibration morphism from Q to P.

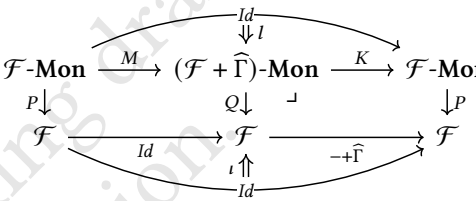
5.2 Modular Models

Now we have enough machinery to define *modular models* of theories on monoids.

Definition 5.4. Given a monoidal theory family $\mathcal{F}$ over some $\mathscr{E}$, a *modular model* $\langle M, l\rangle$ of some theory $\widehat{\Gamma} \in \mathcal{F}$ consists of a functor $M : \mathcal{F}$-**Mon** $\to (\mathcal{F} + \widehat{\Gamma})$-**Mon** and a natural transformation $l : Id \to K \circ M$ such that

(i) $\langle M, Id\rangle$ is a split fibration morphism from $P : \mathcal{F}$-**Mon** $\to \mathcal{F}$ to $Q : (\mathcal{F} + \widehat{\Gamma})$-**Mon** $\to \mathcal{F}$, and

(i) $\langle l, \iota\rangle$ is a split fibration 2-cell $\langle Id, Id\rangle \to \langle K \circ M, - + \widehat{\Gamma}\rangle$ where $\iota : Id \to (- + \widehat{\Gamma})$ is the coprojection.

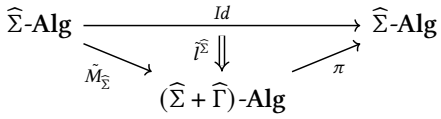


We now give some intuition for the definition. The functor part M transforms existing models $\langle N, \alpha\rangle$ of any $\widehat{\Sigma} \in \mathcal{F}$ into a model $M\langle \widehat{\Sigma}, N, \alpha\rangle$ of the combined theory $\widehat{\Sigma} + \widehat{\Gamma}$, and the new model is related to the original model N by a $\widehat{\Sigma}$-algebra homomorphism $l_{\langle \widehat{\Sigma}, N, \alpha\rangle} : N \to M\langle \widehat{\Sigma}, N, \alpha\rangle$. The intuition can be precisely expressed as the following lemma, which follows from the standard correspondence between the 2-categories of split fibrations and **CAT**-valued functors.

Lemma 5.5. *A modular model of $\widehat{\Gamma} \in \mathcal{F}$ is equivalently a natural family $\tilde{M}_{\widehat{\Sigma}}$ of functors (i.e. a natural transformation between two functors $\mathcal{F} \to$ **CAT**):*

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

and a family $\tilde{l}^{\widehat{\Sigma}}$ of natural transformations for each $\widehat{\Sigma} \in \mathcal{F}$, (in other words, $\tilde{l}$ is a modification [Jacobs 1999]):



*where π is the projection, and $\tilde{l}$ shall make the following hold for all translations $T : \widehat{\Gamma}$-**Alg** $\to \widehat{\Sigma}$-**Alg** and $\langle A, \alpha\rangle \in \widehat{\Gamma}$-**Alg**:*

$$\tilde{l}^{\widehat{\Sigma}}_{T\langle A,\alpha\rangle} = T\tilde{l}^{\widehat{\Gamma}}_{\langle A,\alpha\rangle} : T\langle A, \alpha\rangle \to \tilde{M}_{\widehat{\Sigma}}(T\langle A, \alpha\rangle)$$

Now we give examples of modular models in the context of computational effects. After a trivial example, a modular model for *exception throwing* in the family **ALG**$(\mathscr{E})$ of *algebraic operations* is given in detail (Example 5.7), followed by a general theorem obtaining modular models for theories in **ALG**$(\mathscr{E})$ from *monoid transformers* (Theorem 4). Then

we move on to *scoped operations*, and show an example for exception catching (Example 5.8), also followed by a similar theorem that constructs modular models for scoped operations from *functorial monoid transformers* (Theorem 5).

Example 5.6. For a trivial example, let $\mathcal{F}$ be the monoidal theory family containing only the theory **Mon** itself with the identity translation. In this case, $\mathcal{F}$-**Mon** is just the category $\mathbf{Mon}(\mathscr{E})$ of monoids $\mathscr{E}$ with no additional operations, and a modular model of **Mon** is precisely a (covariant) *monoid transformer* [Jaskelioff and Moggi 2010]:

$$\mathbf{Mon}(\mathscr{E}) \quad \begin{array}{c} \xrightarrow{Id} \\ \Downarrow l \\ \xrightarrow[M]{} \end{array} \quad \mathbf{Mon}(\mathscr{E})$$

Thus monoid transformers (particularly, monad transformers) are special cases of modular models.

Example 5.7. A modular model for the theory $\mathbf{Exc}_E$ of *exception throwing* (Example 3.3) in the family $\mathbf{ALG}(\mathscr{E})$ of algebraic operations for $\mathscr{E} = \mathbf{Endo}_f(\mathbf{Set})$ can be given by the following $\langle \tilde{M}, \tilde{l} \rangle$. Recall that each $\widehat{\Sigma} \in \mathbf{ALG}(\mathscr{E})$ is

$$\widehat{\Sigma} = (S \circ -)\text{-}\mathbf{Mon} \uplus (G \vdash L = R)$$

for some $S, G \in \mathbf{Endo}_f(\mathbf{Set})$, so objects of $\widehat{\Sigma}$-**Alg** are tuples

$$\langle A \in \mathscr{E},\ \alpha : S \circ A \to A,\ \eta^A : I \to A,\ \mu^A : A \circ A \to A \rangle$$

The functor $\tilde{M}_{\widehat{\Sigma}} : \widehat{\Sigma}\text{-}\mathbf{Alg} \to (\widehat{\Sigma} + \mathbf{Exc}_E)\text{-}\mathbf{Alg}$ maps each object $\langle A, \alpha, \eta^A, \mu^A \rangle$ to a $(\widehat{\Sigma} + \mathbf{Exc}_E)$-algebra carried by

$$C_A = A(E + -) = A \circ (E + I) : \mathbf{Set} \to \mathbf{Set} \tag{14}$$

where the second E denotes the constant functor mapping to E, and I is the identity functor. The carrier is equipped with the following structure map

$$\begin{aligned} &[\alpha^\sharp, \beta] : (S \circ C_A) + E \to C_A \quad \text{where} \\ &\alpha^\sharp = (s : S, a : A, e : E + I \vdash (\alpha(s, a), e) : C_A) \\ &\beta = (e : E \vdash (\eta^A, (\mathsf{inj}_1 e)) : C_A) \end{aligned}$$

and with the following monad structure

$$\begin{aligned} \eta^C &= (\cdot \vdash (\eta^A, \mathsf{inj}_2(*)) : C_A) \\ \mu^C &= (a : A,\ e : E + I,\ a' : A,\ e' : E + I \vdash \\ &\qquad \mathsf{let}\ (a'', e'') = d(e, a')\ \mathsf{in}\ (\mu^A(a, a''), \mu^{E+I}(e'', e'))) \end{aligned} \tag{15}$$

where $d : (E + I) \circ A \to A \circ (E + I)$ is a distributive law:

$$\begin{aligned} e : E + I, a : A \vdash \mathsf{case}\ e\ \mathsf{of}\ \{\ &\mathsf{inj}_1\, e' \mapsto (\eta^A, \mathsf{inj}_1 e'); \\ &\mathsf{inj}_2 * \mapsto (a, \mathsf{inj}_2 *)\} \end{aligned}$$

where we use the syntax case e of $\{\cdots\}$ for eliminating coproducts, and μ^{E+I} is the multiplication of the *exception monad* $E + I$:

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

The arrow mapping of $\tilde{M}$ sends $\widehat{\Sigma}$-homomorphism $h : A \to B$ to $h \circ (E + I) : C_A \to C_B$. We also need to show that $\tilde{M}\langle A, \alpha \rangle$ satisfies the equations of $\widehat{\Sigma} + \mathbf{Exc}_E$, but we prove it in a more general setting later. The other component $\tilde{l}$ of the modular model assigns to each $\langle A, \alpha \rangle \in \widehat{\Sigma}\text{-}\mathbf{Alg}$ an arrow $\widehat{\Sigma}$-homomorphism: $a : A \vdash (a, \mathsf{inj}_2 *) : A \circ (E + I)$.

The main ingredient for the modular model in the last example is the well known *exception monad transformer* $C_A = A \circ (E + I)$, but the modular model has additional structure for a model β of the theory $\mathbf{Exc}_E$ and a lifting $\alpha^\sharp$ for existing algebraic operations α. The following is a general result for obtaining modular models of any theory $\widehat{\Sigma} \in \mathbf{ALG}(\mathscr{E})$ from monoid transformers that are equipped with $\widehat{\Sigma}$-algebras.

Theorem 4. *For each $\langle \widehat{\Gamma}, T_\Gamma \rangle \in \mathbf{ALG}(\mathscr{E})$ over some suitable monoidal category $\mathscr{E}$, a functor $H : \mathbf{Mon}(\mathscr{E}) \to \widehat{\Gamma}\text{-}\mathbf{Alg}$ and a natural transformation $\tau : Id \to T_\Gamma \circ H$*

$$\begin{array}{ccccc} & & \widehat{\Gamma}\text{-}\mathbf{Alg} & & \\ & \overset{H}{\nearrow} & \Uparrow \tau & \overset{T_\Sigma}{\searrow} & \\ \mathbf{Mon}(\mathscr{E}) & & \xrightarrow[Id]{} & & \mathbf{Mon}(\mathscr{E}) \end{array}$$

can be extended to a modular model $\langle M, l \rangle$ of $\widehat{\Gamma}$ such that the following diagram commutes and $l_{\langle \langle \widehat{\Sigma}, T_\Sigma \rangle, A, \alpha \rangle} = \tau_{\langle A, T_\Sigma \alpha \rangle}$

$$\begin{array}{ccc} \mathcal{F}\text{-}\mathbf{Mon} & \xrightarrow{M} & (\mathcal{F} + \widehat{\Gamma})\text{-}\mathbf{Mon} \\ \downarrow & & \downarrow \\ \mathbf{Mon}(\mathscr{E}) & \xrightarrow[H]{} & \widehat{\Gamma}\text{-}\mathbf{Alg} \end{array} \tag{16}$$

where the vertical arrows are the projection functors.

This theorem has a wide range of applications. The *state monad transformer* $A \mapsto (A(S \times -))^S$ together with its canonical model [Liang et al. 1995] for the theory $\mathbf{St}_\mathbf{S}$ of *mutable state* (Example 3.5) yields a modular model of $\mathbf{St}_\mathbf{S}$ in $\mathbf{ALG}(\mathbf{Endo}_f(\mathscr{C}))$. The *list monad transformer* $A \mapsto \mu X.A(1 + (- \times X))$ with its canonical model for the theory of *nondeterminism* [Jaskelioff and Moggi 2010] also gives rise to a modular model. Due to space constraints, we do no expand on these examples and move on to scoped operations.

Example 5.8. The theory of *exception throwing and catching* in Example 3.4 is in the family $\mathbf{SCP}(\mathbf{Endo}_f(\mathbf{Set}))$ of scoped operations on finitary monads, and a modular model for it can be constructed if we can extend the modular model of exception throwing in Example 5.7 with (i) a model for catching and (ii) a way to lift existing scoped operations.

(i) To equip the carrier $C_A = A \circ (1 + I)$ (14) with a model $\mathit{catch} : (C_A \times C_A) \circ C_A \to C_A$, we fix E in (14) to be the singleton set 1, since we have assumed only one kind of exception to be caught. Denote by s the following canonical strength in $\mathbf{Endo}_f(\mathbf{Set})$:

$$C_A \times C_A = (A \circ (1{+}I)) \times C_A \to A \circ ((1{+}I) \times C_A) \cong A \circ (C_A{+}I \times C_A)$$

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

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

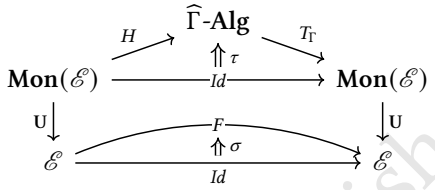
where $\mu^C : C_A \circ C_A \to C_A$ is defined as in (15).

(ii) To lift an existing scoped operation $\alpha : S \circ A \circ A \to A$ to C_A, we define $\alpha^\sharp : S \circ C_A \circ C_A \to C_A$ by the following

$$s : S, a : A, m : 1 + I, k : C_A \vdash \mu^C(\alpha(s, a, \eta^A), k) : C_A$$

Similar to Theorem 4, modular models for scoped operations $\mathbf{SCP}(\mathscr{E})$ can also be obtained from monoid transformers with some more data. The following theorem is crucially based on Jaskelioff and Moggi [2010]'s result that that scoped operations (which they call *first-order operations*) can be lifted along functorial monoid transformers.

Theorem 5. *For each $\langle \widehat{\Gamma}, T_\Gamma \rangle \in \mathbf{SCP}(\mathscr{E})$ over some suitable closed monoidal category $\mathscr{E}$, a functor $H : \mathbf{Mon}(\mathscr{E}) \to \widehat{\Gamma}\text{-}\mathbf{Alg}$ and a natural transformation $\tau : Id \to T_\Gamma \circ H$ such that there is some $F : \mathscr{E} \to \mathscr{E}$ and $\sigma : Id \to F$ such that $\mathbf{U} \circ T_\Gamma \circ H = F \circ \mathbf{U}$ and $\tau \circ \mathbf{U} = \sigma \circ U$*



can be extended to a modular model $\langle M, l \rangle$ of $\widehat{\Gamma}$ such that diagram (16) commutes and $l_{\langle \langle \widehat{\Sigma}, T_\Sigma \rangle, A, \alpha \rangle} = \tau_{\langle A, T_\Sigma \alpha \rangle}$.

In comparison to Theorem 4, the theorem above requires the monoid transformer $\langle T_\Gamma \circ H, \tau \rangle$ to be *functorial*, i.e. being an extension of an endofunctor transformer $\langle F, \sigma \rangle$, because the retract $\epsilon : A/A \to A$ of the Cayley embedding (Example 2.7) used essentially in the proof is *not* a monoid morphism. Many monoid transformers are functorial, including the exception monad transformer underlying Example 5.8, the state monad transformer $A \mapsto (A(S \times -))^S$, and the free monad transformer (also known as the resumption monad transformer) $A \mapsto \mu X.\, A(- + SX)$ for finitary endofunctors $S : \mathscr{C} \to \mathscr{C}$ [Cenciarelli and Moggi 1993].

6 Interpretation with Modular Models

In this section, we explain how modular models can be used for *interpreting* abstract syntax, i.e. initial algebras, of equational theories (Lemma 6.1). We also show how modular models of different theories *compose* (Definition 6.2) and a *fusion theorem* (Theorem 6) saying that interpreting a term with two modular models sequentially is equal to interpreting with the *composite* of the two models.

6.1 Interpretation

Fixing a monoidal theory family $\mathcal{F}$, every theory $\langle \widehat{\Sigma}, T_\Sigma \rangle$ in $\mathcal{F}$ has free algebras and in particular an initial algebra $\langle \widehat{\Sigma}^*, \alpha^\Sigma \rangle$ by Definition 4.1. The initial algebra is mapped by every modular model $\langle M, l \rangle$ of some $\widehat{\Gamma} \in \mathcal{F}$ to $\tilde{M}_{\widehat{\Sigma}} \langle \widehat{\Sigma}^*, \alpha^\Sigma \rangle$ in $(\widehat{\Sigma} + \widehat{\Gamma})$-**Alg** by Lemma 5.5. Then the initial algebra $(\widehat{\Sigma} + \widehat{\Gamma})^*$ induces a unique homomorphism:

$$\tilde{h}_{\widehat{\Sigma}} : (\widehat{\Sigma} + \widehat{\Gamma})^* \to \tilde{M}_{\widehat{\Sigma}} \langle \widehat{\Sigma}^*, \alpha^\Sigma \rangle \tag{17}$$

The intuition is that this morphism *modularly* interprets the $\widehat{\Gamma}$-operations in syntactic terms $(\widehat{\Sigma} + \widehat{\Gamma})^*$ with a modular model M, leaving operations from other theories $\widehat{\Gamma}$ uninterpreted. Specially, if $\widehat{\Sigma}$ is the theory with no operations (which is always in $\mathcal{F}$ since it is the coproduct indexed by the empty set in $\mathcal{F}$ and $\mathcal{F}$ is closed under coproducts), then $\widehat{\Sigma}^*$ is the initial monoid I and $(\widehat{\Sigma} + \widehat{\Gamma})^* \cong \widehat{\Gamma}^*$. The morphism $\tilde{h}_{\widehat{\Sigma}} : \widehat{\Gamma}^* \to \tilde{M} \langle I, \alpha^I \rangle$ simply interprets abstract syntax $\widehat{\Gamma}^*$.

The interpretation morphism (17) is natural in $\widehat{\Sigma}$ in a suitable sense. Define a functor $(-)^\star : \mathcal{F} \to \mathcal{F}$-**Mon** by

$$(\widehat{\Sigma} \in \mathcal{F}) \mapsto \langle \widehat{\Sigma},\ \widehat{\Sigma}^*,\ \alpha^\Sigma : \Sigma(\widehat{\Sigma}^*) \to \widehat{\Sigma}^* \rangle$$
$$(\sigma : \widehat{\Sigma} \to \widehat{\Gamma}) \mapsto \langle \sigma,\ u : \langle \widehat{\Sigma}^*,\ \alpha^\Sigma \rangle \to \sigma \langle \widehat{\Gamma}^*,\ \alpha^\Gamma \rangle \rangle$$

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

Lemma 6.1. *Given a modular model $\langle M, l \rangle$ of $\widehat{\Gamma} \in \mathcal{F}$, there is a natural transformation*

$$h^M : (- + \widehat{\Gamma})^\star \to KM(-)^\star : \mathcal{F} \to \mathcal{F}\text{-}\mathbf{Mon} \tag{18}$$

such that all $h^M_{\widehat{\Sigma}} = \langle id_{\widehat{\Sigma} + \widehat{\Gamma}},\ \tilde{h}_{\widehat{\Sigma}} \rangle$, where K is defined as in (13).

Remark. Since $\widehat{\Sigma} + \widehat{\Gamma}$ extends the theory of monoids, the interpretation (18) is always a monoid morphism, and thus it preserves monoid multiplication μ. This is called the *semantic substitution lemma* [Tennent 1991] for $\mathscr{E} = \mathbf{Set}^{\mathcal{F}}$ since μ stands for variable substitution in this case.

6.2 Composition and Fusion

The point of modular models is to be composable, which can be done by composing the underlying functors.

Definition 6.2. The *composite* $\widehat{M} \triangleright \widehat{N}$ of two modular models $\widehat{M} = \langle M, l^M \rangle$ of $\widehat{\Gamma}$ and $\widehat{N} = \langle N, l^N \rangle$ of $\widehat{\Phi}$ is a modular model of $\widehat{\Gamma} + \widehat{\Phi}$ given by $\langle T, l^N \circ l^M \rangle$ where

$$T : \mathcal{F}\text{-}\mathbf{Mon} \to (\mathcal{F} + \widehat{\Gamma} + \widehat{\Phi})\text{-}\mathbf{Mon}$$

is induced by the pullback property (13) of $(\mathcal{F} + \widehat{\Gamma} + \widehat{\Phi})$-**Mon** and the commutativity of the following diagram:

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

Example 6.3. Let M_E be the modular model of *exception* throwing and catching (Example 5.8), and M_S be the modular model of *mutable state* arising from the state monad transformer [Liang et al. 1995] by Theorem 4. The composite $M_E \triangleright M_S$ is a modular model of the coproduct $\mathbf{TC} + \mathbf{St_S}$ of the theories of exception and mutable state.

Remark 6.4. Although the coproduct of theories is commutative, $\widehat{\Gamma} + \widehat{\Phi} \cong \widehat{\Phi} + \widehat{\Gamma}$, the composition of modular models is *not*. For example, the opposite order $M_S \triangleright M_E$ of composing the modular models in Example 6.3 gives rise to a different modular model of the coproduct $\mathbf{TC} + \mathbf{St_S}$, but state and exception *interact* differently in these two models. Operationally, 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 in different contexts, so the language designer needs to determine the correct order of composing modular models according to the desired interaction.

A composite model $\widehat{N} \triangleright \widehat{M}$ can be used for interpreting $(\widehat{\Sigma} + (\widehat{\Gamma} + \widehat{\Phi}))^\star$ by (18). Since $(\widehat{\Sigma} + (\widehat{\Gamma} + \widehat{\Phi}))^\star \cong ((\widehat{\Sigma} + \widehat{\Gamma}) + \widehat{\Phi})^\star$, it can also be interpreted by using $\widehat{N}$ and $\widehat{M}$ sequentially. The results of two ways can be shown to be equal.

Theorem 6 (Fusion)**.** *Given modular models $\widehat{M}$ of $\widehat{\Gamma}$ and $\widehat{N}$ of $\widehat{\Phi}$, the following diagram commutes:*

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

This theorem allows two consecutive interpretations of terms $(\widehat{\Sigma} + \widehat{\Gamma} + \widehat{\Phi})^\star$ to be *fused* into one with the composite model, eliminating the intermediate result $(\widehat{\Sigma} + \widehat{\Gamma})^\star$. Fusion is used for optimising [Wu and Schrijvers 2015] and reasoning about interpretations [Yang and Wu 2021] of algebraic operations on monads, and the theorem here makes these techniques applicable to all monoidal theory families.

7 Related Work and Conclusion

Effect Handlers. The most closely related work is the line of research on *handlers of algebraic effects* introduced by Plotkin and Pretnar [2009, 2013]. Semantically, handlers are models of first-order equational theories, and are used for interpreting free algebras. As a programming construct, handlers offer a *composable* approach to user-defined (algebraic) effects, essentially relying on the fact that algebraic operations can be lifted canonically. Many implementations of handlers have been developed, both as libraries (e.g. Kammar et al. [2013]) and languages (e.g. Bauer and Pretnar [2015]). In particular, Schrijvers et al. [2019b] introduced *modular handlers* in Haskell, which are handlers polymorphic in the residual effects. Yang and Wu [2021] show that modular handlers can be fused and are suitable for reasoning.

The initial motivation of the present work was to develop a clear categorical formulation of modular handlers, and generalise them to monoids with non-algebraic operations. However, our definition of *modular models* deviates from modular handlers in the way that modular models always have a monoid structure, while handlers may only model the operations. The distinction is analogous to Arkor and Fiore [2020]'s models of *simply typed syntax* versus *simple type theories*. We believe that our deviation is a reasonable choice since (i) many practical examples of modular handler have a monoid structure anyway; and (ii) many theories of non-algebraic operations inherently involve monoid operations in their equational laws, such as Example 3.4, so a handler for such a theory without a monoid structure seems pointless.

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

Monad Transformers. Moggi [1989], Wadler [1990], and Spivey [1990] pioneered using *monads* to model computational effects in functional programming languages. Liang et al. [1995] introduced *monad transformers* in Haskell to overcome the problem that monads do not straightforwardly compose. Their implementation also includes *type classes* of monads with certain operations, which can be seen as a Haskell realisation of algebraic theories, but they need to lift existing operations along monad transformers in ad-hoc ways. Later, Jaskelioff [2009] showed how to lift $\hat{\Sigma}$-operations along *functorial monad transformers*, and this result is generalised to *monoid transformers* by Jaskelioff and Moggi [2010]. The present work is a conceptual development of monoid transformers, bringing equational theories and free algebras into the framework. In our view, *monoid transformers are models of computational effects, rather than effects themselves.*

Theories and Syntax. *Algebraic theories* and their connections to *Lawvere theories* and *monads* have been studied for decades (see e.g. [Adamek et al. 2010]). In this paper, we use Fiore and Hur [2009]'s *equational systems* to define equational theories for their generality and simplicity. *Abstract syntax* can be modelled as initial algebras of theories [Goguen et al. 1977], and Fiore et al. [1999] show that abstract syntax with variable bindings can be modelled as initial algebras in a presheaf category, and they introduce Σ-*monoids*, which play a central role in the present paper. Their work leads to the line of research on *second-order algebraic theories*

[Fiore 2008; Fiore and Hur 2010; Fiore and Mahmoud 2010; Fiore and Szamozvancev 2022]. Ghani et al. [2006] also model the syntax of *explicit substitution* in a functorial category, which Piróg et al. [2018] base on to model scoped operations. In comparison, the focus of the present work is (modular) models of abstract syntax, rather than syntax itself.

Conclusion. We have developed a categorical framework of modular models of equational theories that are freely composable. We applied this framework to families of algebraic and scoped operations on monoids, bridging algebraic theories and monoid transformers. As future work, we can consider variations of modular models that are not covariant in their domains, which will encompass modular models based on the continuation monad transformer. Another important direction is operational semantics and type systems for monoidal theory families and modular models.

References

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

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

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

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

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

Richard S. Bird. 1987. An Introduction to the Theory of Lists. In *Logic of Programming and Calculi of Discrete Design*, Manfred Broy (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 5–42. https://doi.org/10.1007/978-3-642-87374-4_1

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

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

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

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

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

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

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

Marcelo Fiore and Chung-Kil Hur. 2010. Second-Order Equational Logic (Extended Abstract). In *Computer Science Logic*, Anuj Dawar and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 320–335.

Marcelo Fiore and Ola Mahmoud. 2010. Second-Order Algebraic Theories. In *Mathematical Foundations of Computer Science 2010*, Petr Hliněný and Antonín Kučera (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 368–380.

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

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

Marcelo P. Fiore and Daniele Turi. 2001. Semantics of Name and Value Passing. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Lambert Meertens. 1986. Algorithmics : towards programming as a mathematical activity. In *Towards programming as a mathematical activity. Mathematics and computer science*. 289–334.

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

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

Koki Nishizawa and John Power. 2009. Lawvere theories enriched over a general base. *Journal of Pure and Applied Algebra* 213, 3 (2009), 377–386. https://doi.org/10.1016/j.jpaa.2008.07.009

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

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

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

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

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

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

Tom Schrijvers, Maciej Piróg, Nicolas Wu, and Mauro Jaskelioff. 2019a. Monad transformers and modular algebraic effects: what binds them together. In *Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell, Haskell@ICFP 2019, Berlin, Germany, August 18-23, 2019*. 98–113. https://doi.org/10.1145/3331545.3342595

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

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

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

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

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

Birthe van den Berg, Tom Schrijvers, Casper Bach Poulsen, and Nicolas Wu. 2021. Latent Effects for Reusable Language Components. In *Programming Languages and Systems*, Hakjoo Oh (Ed.). Springer International Publishing, Cham, 182–201.

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

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

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

Nicolas Wu and Tom Schrijvers. 2015. Fusion for Free. In *Mathematics of Program Construction*, Ralf Hinze and Janis Voigtländer (Eds.). Springer International Publishing, Cham, 302–322. https://doi.org/978-3-319-19797-5_15

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. *To appear in ESOP 2022* (2022).

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

A Omitted Proofs

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

Lemma 3.8. *(i) When $\mathscr{C}$ has all (small) coproducts, so does the category* $\mathbf{Eqs}(\mathscr{C})$. *(ii) When $\mathscr{C}$ has binary coproducts and there is a reflection* Σ-**Alg** $\rightleftarrows$ $\widehat{\Sigma}$-**Alg**, *then all pairs of translations* $T_1, T_2 : \widehat{\Gamma} \to \widehat{\Sigma}$ *has coequalisers.*

Proof sketch of Lemma 3.8. (i) The coproduct of a set of equational systems is obtained by taking the coproduct of signatures and equations. Precisely, the coproduct $\coprod_{i\in I} \widehat{\Sigma}_i$ has signature $\coprod_i \Sigma_i$ and equation $\coprod_{i\in I} \Gamma_i \vdash L = R$ where

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

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

(ii) The codomain $\widehat{\Sigma}'$ of the coequaliser is $\widehat{\Sigma}$ extended with an additional equation of the following two functorial terms

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

The coequaliser $\widehat{\Sigma} \to \widehat{\Sigma}'$ is the inclusion functor:

$$\widehat{\Sigma}'\text{-}\mathbf{Alg} \hookrightarrow \widehat{\Sigma}\text{-}\mathbf{Alg}.$$

□

Theorem 1. *The full subcategory* $\mathbf{Eqs}_\omega(\mathscr{C}) \subseteq \mathbf{Eqs}(\mathscr{C})$ *of equational systems whose functorial signature and context preserve colimits of ω-chains is cocomplete if $\mathscr{C}$ is cocomplete.*

Proof of Theorem 1. For such equational systems $\widehat{\Sigma}$, there are reflections Σ-**Alg** $\rightleftarrows$ $\widehat{\Sigma}$-**Alg** by Proposition 3.2. Then cocompleteness follows from the lemma above since colimits can be constructed using coequalisers and coproducts. □

Theorem 2. *For a monoidal cocomplete category $\mathscr{E}$ with $\square : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ preserving coproducts in the first argument and colimits of ω-chains in both arguments, there is an equivalence* $\mathbf{Mon}(\mathscr{E}) \cong \mathbf{ALG}(\mathscr{E})$ *of categories where* $\mathbf{Mon}(\mathscr{E})$ *is the category of monoids in $\mathscr{E}$.*

Proof of Theorem 2. The direction $\mathbf{ALG}(\mathscr{E}) \to \mathbf{Mon}(\mathscr{E})$ of the equivalence sends every theory $\widehat{\Sigma}$ in $\mathbf{ALG}(\mathscr{E})$ to its initial algebra $\langle \widehat{\Sigma}^*, \alpha^\Sigma \rangle$, which is a monoid since all theories in $\mathbf{ALG}(\mathscr{E})$ extend the theory of monoids. To extend the mapping to a functor, every translation $T : \widehat{\Sigma} \to \widehat{\Gamma} \in \mathbf{ALG}(\mathscr{E})$ is mapped to the unique $\widehat{\Sigma}$-homomorphism out of the initial algebra:

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

which is also a monoid morphism from $\widehat{\Sigma}^*$ to $\widehat{\Gamma}^*$ since T preserves monoid operations by the definition of $\mathbf{ALG}(\mathscr{E})$.

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

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

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

saying that $\mathrm{op} : M \square \tau \to \tau$ is a monoid action on τ. Every monoid morphism $f : \widehat{M} \to \widehat{N}$ is mapped to the translation $\widehat{M}$-**Act** $\to \widehat{N}$-**Act** sending $\widehat{N}$-actions (note the contravariance of translations)

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

to $\widehat{M}$-actions

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

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

- Starting from a monoid $\widehat{M}$, it can be shown that the category $(\widehat{M}\text{-}\mathbf{Act})\text{-}\mathbf{Alg}$ is just the coslice category $\widehat{M}/\mathbf{Mon}(\mathscr{E})$. Thus the initial algebra of $\widehat{M}$-**Act** is $\widehat{M}$ as required.
- Starting from a theory $\widehat{\Sigma} \in \mathbf{ALG}(\mathscr{E})$, it is mapped to the monoid $\widehat{\Sigma}^*$, which is then mapped to the theory $\widehat{\Sigma}^*$-**Act**. We need to construct a isomorphism translation $T : \widehat{\Sigma} \to \widehat{\Sigma}^*$-**Act**. Given a monoid A with $\alpha : \widehat{\Sigma}^* A \to A$ satisfying the laws of $\widehat{\Sigma}^*$-**Act**, T maps it to

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

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

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

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

□

Theorem 3. *Let $\mathscr{E}$ be a monoidal cocomplete category with $\square : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ preserving coproducts in the first argument and colimits of ω-chains in both arguments,* $\mathbf{ALG}(\mathscr{E})$ *is a coreflective subcategory of* $\mathbf{Mon}/\mathbf{Eqs}_\omega(\mathscr{E})$*:*

$$\mathbf{Mon}(\mathscr{E}) \underset{\cong}{\overset{\longleftarrow}{\longrightarrow}} \mathbf{ALG}(\mathscr{E}) \underset{\top}{\overset{\longleftarrow}{\hookrightarrow}} \mathbf{Mon}/\mathbf{Eqs}_\omega(\mathscr{E})$$

Proof sketch of Theorem 3. Every theory $\widehat{\Sigma} \in \mathbf{Mon}/\mathbf{Eqs}_\omega(\mathscr{E})$ always has an initial algebra carried by $\widehat{\Sigma}^* \in \mathscr{E}$ by Theorem 1, and $\widehat{\Sigma}^*$ is a monoid since $\widehat{\Sigma}$ comes with a translation $\mathbf{Mon} \to \widehat{\Sigma}$. The coreflector $\lfloor - \rfloor$ maps the theory $\widehat{\Sigma}$ to $\widehat{\Sigma}^*$-**Act** $\in \mathbf{ALG}(\mathscr{E})$ as in the proof of Theorem 2. For every theory $\widehat{\Gamma} \in \mathbf{ALG}(\mathscr{E})$, it can be shown that each translation in the hom-set $\mathbf{Mon}/\mathbf{Eqs}_\omega(\widehat{\Gamma}, \widehat{\Sigma})$ is equivalently a monoid morphism $\widehat{\Gamma}^* \to \widehat{\Sigma}^*$, which is also equivalently a translation in $\mathbf{ALG}(\widehat{\Gamma}, \lfloor \widehat{\Sigma} \rfloor)$ by Theorem 2. □

Proposition 4.2. *The family* $\mathbf{ALG}(\mathscr{E})$ *of algebraic operations is a coreflective subcategory of* $\mathbf{SCP}(\mathscr{E})$*, thus:*

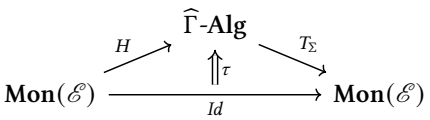
$$\mathscr{E} \underset{\perp}{\overset{}{\rightleftarrows}} \mathbf{Mon}(\mathscr{E}) \underset{\cong}{\overset{}{\rightleftarrows}} \mathbf{ALG}(\mathscr{E}) \underset{\perp}{\overset{}{\rightleftarrows}} \mathbf{SCP}(\mathscr{E})$$

Proof of Proposition 4.2. This directly follows from Theorem 3 by restricting the coreflector to $\mathbf{SCP}(\mathscr{E})$ and the fact that $\mathbf{ALG}(\mathscr{E}) \subseteq \mathbf{SCP}(\mathscr{E})$. □

Proposition 4.3. *The family* $\mathbf{ALG}(\mathbf{Set}^{\mathcal{F}})$ *of algebraic operations is a coreflective subcategory of* $\mathbf{VAR}(\mathbf{Set}^{\mathcal{F}})$*.*

Proof of Proposition 4.3. This directly follows from Theorem 3 by restricting the coreflector to $\mathbf{SCP}(\mathscr{E})$ and the fact that $\mathbf{ALG}(\mathscr{E}) \subseteq \mathbf{VAR}(\mathscr{E})$. □

Theorem 4. *For each* $\langle \widehat{\Gamma}, T_\Gamma \rangle \in \mathbf{ALG}(\mathscr{E})$ *over some suitable monoidal category* $\mathscr{E}$*, a functor* $H : \mathbf{Mon}(\mathscr{E}) \to \widehat{\Gamma}\text{-}\mathbf{Alg}$ *and a natural transformation* $\tau : Id \to T_\Gamma \circ H$



can be extended to a modular model $\langle M, l \rangle$ *of* $\widehat{\Gamma}$ *such that the following diagram commutes and* $l_{\langle\langle \widehat{\Sigma}, T_\Sigma \rangle, A, \alpha \rangle} = \tau_{\langle A, T_\Sigma \alpha \rangle}$

$$\begin{array}{ccc} \mathcal{F}\text{-}\mathbf{Mon} & \xrightarrow{M} & (\mathcal{F} + \widehat{\Gamma})\text{-}\mathbf{Mon} \\ \downarrow & & \downarrow \\ \mathbf{Mon}(\mathscr{E}) & \xrightarrow[H]{} & \widehat{\Gamma}\text{-}\mathbf{Alg} \end{array} \tag{16}$$

where the vertical arrows are the projection functors.

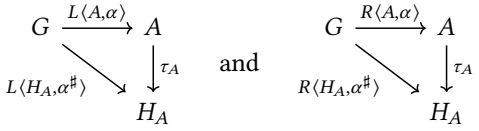
Proof sketch of Theorem 4. For each object $\langle\langle \widehat{\Sigma}, T_\Sigma \rangle, A, \alpha \rangle$ of $\mathcal{F}$-**Mon** with

$$\widehat{\Sigma} = (S \circ -)\text{-}\mathbf{Mon} \uplus (G \vdash L = R)$$

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

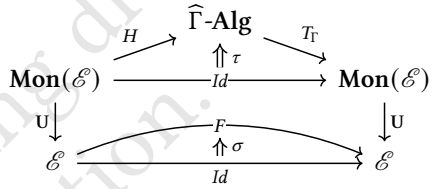
$$\alpha^\sharp = (s : S, h : H_A \vdash \mu^H(\tau_A(\alpha_S(s, \eta^A)),\ h) : H_A) \tag{19}$$

where H_A and τ_A stand for the carrier of $H\langle A, T_\Sigma \alpha \rangle$ and $\tau_{\langle A, T_\Sigma \alpha \rangle} : A \to H_A$ respectively, and $\alpha_S : S \circ A \to A$ is the component of α for the algebraic operation on A. We also need to show that the operation (19) satisfies the equation $G \vdash L = R$. This follows from the functoriality of $L, R : (S \circ - + \Sigma_{\mathbf{Mon}})\text{-}\mathbf{Alg} \to G\text{-}\mathbf{Alg}$, which implies that the following diagrams commute



If α satisfies the equation $L = R$ (i.e. $L\langle A, \alpha \rangle = R\langle A, \alpha \rangle$), so does $\alpha^\sharp$. It can be checked that the above M (with an evident arrow mapping) and l defined as $l_{\langle\langle \widehat{\Sigma}, T_\Sigma \rangle, A, \alpha \rangle} = \tau_{\langle A, T_\Sigma \alpha \rangle}$ satisfy the conditions for being a modular model. □

Theorem 5. *For each* $\langle \widehat{\Gamma}, T_\Gamma \rangle \in \mathbf{SCP}(\mathscr{E})$ *over some suitable closed monoidal category* $\mathscr{E}$*, a functor* $H : \mathbf{Mon}(\mathscr{E}) \to \widehat{\Gamma}\text{-}\mathbf{Alg}$ *and a natural transformation* $\tau : Id \to T_\Gamma \circ H$ *such that there is some* $F : \mathscr{E} \to \mathscr{E}$ *and* $\sigma : Id \to F$ *such that* $\mathbf{U} \circ T_\Gamma \circ H = F \circ \mathbf{U}$ *and* $\tau \circ \mathbf{U} = \sigma \circ U$



can be extended to a modular model $\langle M, l \rangle$ *of* $\widehat{\Gamma}$ *such that diagram (16) commutes and* $l_{\langle\langle \widehat{\Sigma}, T_\Sigma \rangle, A, \alpha \rangle} = \tau_{\langle A, T_\Sigma \alpha \rangle}$*.*

Proof sketch of Theorem 5. Compared to Theorem 4, what is essentially new is how scoped operations $\alpha : S \circ A \circ A \to A$ on existing monoids $\langle A, \eta^A, \mu^A \rangle$ are lifted to $\mathbf{U}(T_\Gamma(H\langle A, \eta^A, \mu^A \rangle)) = FA$. The lifting given by Jaskelioff and Moggi [2010] can be denoted as follows, which makes use of the Cayley embedding of monoids $A \to A/A$ (Example 2.7):

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

It can be showed that this lifting preserves functorial equations $E \vdash L = R$ satisfied by α with constant contexts E by the same argument as for Theorem 4. □

Theorem 6 (Fusion)**.** *Given modular models* $\widehat{M}$ *of* $\widehat{\Gamma}$ *and* $\widehat{N}$ *of* $\widehat{\Phi}$*, the following diagram commutes:*

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

Proof sketch of Theorem 6. It is a consequence of the initiality of $(\widehat{\Sigma} + \widehat{\Gamma} + \widehat{\Phi})^*$. □