RESEARCH-ARTICLE

Scoped Effects, Scoped Operations, and Parameterized Algebraic Theories

CRISTINA MATACHE, The University of Edinburgh, Edinburgh, Scotland, U.K.

SAM LINDLEY, University of Birmingham, Birmingham, West Midlands, U.K.

SEAN K MOSS, University of Birmingham, Birmingham, West Midlands, U.K.

SAM STATON, University of Oxford, Oxford, Oxfordshire, U.K.

NICOLAS WU, Imperial College London, London, U.K.

ZHIXUAN YANG, Imperial College London, London, U.K.

Published: 16 June 2025
Online AM: 05 May 2025
Accepted: 04 March 2025
Revised: 29 December 2024
Received: 27 April 2024

Scoped Effects, Scoped Operations, and Parameterized Algebraic Theories

CRISTINA MATACHE and SAM LINDLEY, University of Edinburgh, Edinburgh, United Kingdom of Great Britain and Northern Ireland
SEAN MOSS, University of Birmingham, Birmingham, United Kingdom of Great Britain and Northern Ireland
SAM STATON, University of Oxford, Oxford, United Kingdom of Great Britain and Northern Ireland
NICOLAS WU and ZHIXUAN YANG, Imperial College London, London, United Kingdom of Great Britain and Northern Ireland

Notions of computation can be modeled by monads. *Algebraic effects* offer a characterization of monads in terms of algebraic operations and equational axioms, where operations are basic programming features, such as reading or updating the state, and axioms specify observably equivalent expressions. However, many useful programming features depend on additional mechanisms such as delimited scopes or dynamically allocated resources. Such mechanisms can be supported via extensions to algebraic effects including *scoped effects* and *parameterized algebraic theories*. We present a fresh perspective on scoped effects by translation into a variation of parameterized algebraic theories. The translation enables a new approach to equational reasoning for scoped effects and gives rise to an alternative characterization of monads in terms of generators and equations involving both scoped and algebraic operations. We demonstrate the power of our approach by way of equational characterizations of several known models of scoped effects.

CCS Concepts: • **Theory of computation → Denotational semantics**; **Categorical semantics**;

Additional Key Words and Phrases: algebraic effects, scoped effects, monads, category theory, algebraic theories

ACM Reference format:
Cristina Matache, Sam Lindley, Sean Moss, Sam Staton, Nicolas Wu, and Zhixuan Yang. 2025. Scoped Effects, Scoped Operations, and Parameterized Algebraic Theories. *ACM Trans. Program. Lang. Syst.* 47, 2, Article 8 (June 2025), 33 pages.
https://doi.org/10.1145/3731678

This work was supported by the UKRI Future Leaders Fellowship "Effect Handler Oriented Programming" (reference number MR/T043830/1), ERC Project BLAST, and AFOSR Award No. FA9550-21-1-003.
Authors' Contact Information: Cristina Matache (corresponding author), University of Edinburgh, Edinburgh, United Kingdom of Great Britain and Northern Ireland; e-mail: cristina.matache@ed.ac.uk; Sam Lindley, University of Edinburgh, Edinburgh, United Kingdom of Great Britain and Northern Ireland; e-mail: sam.lindley@ed.ac.uk; Sean Moss, University of Birmingham, Birmingham, United Kingdom of Great Britain and Northern Ireland; e-mail: s.k.moss@bham.ac.uk; Sam Staton, University of Oxford, Oxford, United Kingdom of Great Britain and Northern Ireland; e-mail: sam.staton@cs.ox.ac.uk; Nicolas Wu, Imperial College London, London, United Kingdom of Great Britain and Northern Ireland; e-mail: n.wu@imperial.ac.uk; Zhixuan Yang, Imperial College London, London, United Kingdom of Great Britain and Northern Ireland; e-mail: s.yang20@imperial.ac.uk.

ACM 1558-4593/2025/6-ART8
https://doi.org/10.1145/3731678

1 Introduction

The central idea of *algebraic effects* [28] is that impure computation can be built and reasoned about equationally, using an algebraic theory. *Effect handlers* [32] are a way of implementing algebraic effects and provide a method for modularly programming with different effects. More formally, an effect handler gives a model for an algebraic theory. In this article we develop equational reasoning for a notion arising from an extension of handlers, called *scoped effects*, using the framework of *parameterized algebraic theories*.

The central idea of *scoped effects* (Section 2.2) is that certain parts of an impure computation should be dealt with one way, and other parts another way, inspired by scopes in exception handling. Compared to algebraic effects, the crucial difference is that the scope on which a scoped effect acts is delimited. This difference leads to a complex relationship with monadic sequencing ($\gg\!=$). The theory and practice of scoped effects [5, 25, 43–46] has primarily been studied by extending effect handlers to deal with not just algebraic operations, but also more complex scoped operations. They form the basis of the fused-effects and polysemy libraries for Haskell. Aside from exception handling, other applications include back-tracking in parsing [44] and timing analysis in telemetry [42].

Parameterized algebraic theories (Section 2.3) extend plain algebraic theories with variable binding operations for an abstract type of parameters. They have been used to study various resources including logic variables in logic programming [37], channels in the π-calculus [38], code pointers [10], qubits in quantum programming [40], and urns in probabilistic programming [41].

Contributions. We propose an equational perspective for scoped effects where *scopes are resources*, by analogy with other resources like file handles. We develop this perspective using the framework of *parameterized algebraic theories*, which provides an algebraic account of effects with resources and instances. We realize scoped effects by encoding the scopes as resources with open/close operations, analogous to opening/closing files. This article provides:

- —the first syntactic sound and complete equational reasoning system for scoped effects, based on equational reasoning with parameterized algebraic theories (Propositions 4.3 and 4.6);
- —a canonical notion of semantic model for scoped effects supporting four key examples from the literature: nondeterminism with semi-determinism (Theorem 4.12), catching exceptions (Theorem 4.13), state with local values (Theorem 4.14), and nondeterminism with cut (Theorem 4.16); and
- —a reconstruction of the previous categorical analysis of scoped effects via parameterized algebraic theories: the constructors $(\triangleleft, \triangleright)$ are shown to be not *ad hoc*, but rather the crucial mechanism for arities/coarities in parameterized algebraic theories (Theorem 4.10).

Example: Nondeterminism with Semi-Determinism. We now briefly illustrate the intuition underlying the connection between scoped effects and parameterized algebraic theories through an example. (See Examples 2.3 and 2.7 for further details.) Let us begin with two algebraic operations: $\mathsf{or}(x, y)$, which nondeterministically chooses between continuing as computation x or as computation y, and fail, which fails immediately. We add semi-determinism in the form of a *scoped* operation $\mathsf{once}(x)$, which chooses the first branch of the computation x that does not fail. Importantly, the scope that once acts on is delimited. The left program below returns 1; the right one returns 1 or 2, as the second or is outside the scope of once.

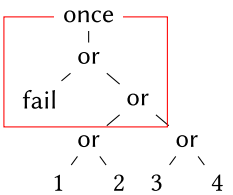

Fig. 1. Illustrating (1).

$$\mathsf{once}(\mathsf{or}(\mathsf{or}(1,2),\mathsf{or}(3,4))) \qquad \mathsf{once}(\mathsf{or}(1,3)) \gg\!= \lambda x.\, \mathsf{or}(x, x+1).$$

Now consider a slightly more involved example, which also returns 1 or 2:

$$\mathsf{once}(\mathsf{or}(\mathsf{fail},\ \mathsf{or}(1,3))) \gg\!= \lambda x.\, \mathsf{or}(x, x+1) \tag{1}$$

depicted as a tree in Figure 1 where the red box delimits the scope of once. We give an encoding of term (1) in a parameterized algebraic theory as follows:

$$\mathsf{once}(a.\mathsf{or}(\mathsf{fail}, \mathsf{or}(\mathsf{close}(a; \mathsf{or}(1,2)), \mathsf{close}(a; \mathsf{or}(3,4))))), \tag{2}$$

where a is the name of the scope opened by once and closed by the special close operation. By equational reasoning for scoped effects (Section 3) and the equations for nondeterminism (Figure 4), we can prove that the term (2) is equivalent to $\mathsf{or}(1,2)$.

Aside. This continuation-passing style notation is natural for algebraic effects viewed as algebraic theories, but when programming, one often uses equivalent direct-style *generic effects* [30] such as $\underline{\mathsf{unknown}} : \mathsf{unit} \to \mathsf{bool}$, returning a nondeterministic Boolean, where $\mathsf{or}(x, y)$ can be recovered by pattern matching on the result of $\underline{\mathsf{unknown}}\ ()$. See (8).

Changes from the Conference Version. This article is an extended version of a "fresh perspective" short paper published at ESOP 2024 [17]. The additions include the following:

- —the scoped effect of explicit nondeterminism with cut treated as a parameterized theory (Example 3.9) and its free model (Section 4.4.4);
- —a discussion about how monads support operations (Definitions 2.11 and 2.12), leading to a comparison between models of algebraic theories and models of parameterized theories in Propositions 4.8 and 4.9;
- —a new section about constructing a parameterized theory from an arbitrary family of scoped operations (Section 4.5) and how their models are related (Theorem 4.17);
- —proofs of the general theorems about models from Section 4.2, as well as proofs of freeness for the examples of models from Section 4.4. The proofs of freeness involve exhibiting normal forms for each parameterized theory;
- —an expanded discussion of existing work about the higher-order syntax approach to scoped effects, in Section 2.2;
- —the observation that one of the equations in the parameterized theory on nondeterminism with once is actually derivable from the others (Example 3.5);
- —more details about the alternative definition of catching exceptions as a parameterized theory (Example 3.7); and
- —a discussion of future research about combinations of scoped theories in Section 5.

2 Background

In this section we put the present work in context and recall some basic concepts useful for the technical development in Sections 3 and 4.

First, in Section 2.1 we discuss some features and limitations of Perspective 2.1 (Plotkin and Power [29]) which identifies a notion of effectful computation with an ordinary (first-order) algebraic theory. Then, in Section 2.2 we compare some existing perspectives on *scoped effects*, which are designed to generalize algebraic effects. In particular we consider Perspective 2.10 (Wu et al. [44]), which explains scoped operations in terms of the semantics of an ordinary algebraic theory, as operations which need not commute with monadic sequencing.

Finally, in Section 2.3 we introduce our new Perspective 2.15 which says that scoped operations are operations that allocate and consume a certain resource: names of scopes. We give some background on the *parameterized algebraic theories* which we will use to give our perspective a formal syntactic framework in Section 3.

2.1 Algebraic Effects

Moggi [22, 23] shows that many non-pure features of programming languages, typically referred to as *computational effects*, can be modeled uniformly as *monads*, but the question is—*how do we construct a monad for an effect*, or putting it differently, *where do the monads modeling effects come from*? A classical result in category theory is that finitary monads over the category of sets are equivalent to *algebraic theories* [16, 18]: An algebraic theory gives rise to a finitary monad by the free-algebra construction, and conversely every finitary monad is presented by a certain algebraic theory. Motivated by this correspondence, Plotkin and Power [29] show that many monads that are used for modeling computational effects can be presented by algebraic theories of some basic effectful operations and some computationally natural equations. This observation led them to the following influential perspective on computational effects [29], which is nowadays commonly referred to as *algebraic effects*.

Perspective 2.1 (Plotkin and Power [29]). *An effect is realized by an algebraic theory of its basic operations, so it* determines *a monad but is not identified with the monad.*

We review the framework of algebraic effects in the simplest form here and refer the reader to Plotkin and Power [31] and Bauer [2] for more discussion.

Definition 2.2. A (first-order finitary) *algebraic signature* $\Sigma = \langle |\Sigma|, ar \rangle$ consists of a set $|\Sigma|$, whose elements are referred to as *operations*, together with a mapping $ar : |\Sigma| \to \mathbb{N}$, associating a natural number to each operation, called its *arity*.

Given a signature $\Sigma = \langle |\Sigma|, ar \rangle$, we will write $O : n$ for an operation $O \in |\Sigma|$ with $ar(O) = n$. The *terms* $\mathsf{Tm}_\Sigma(\Gamma)$ in a context Γ, which is a finite list of variables, are inductively generated by

$$\frac{}{\Gamma, x, \Gamma' \vdash x} \qquad \frac{O : n \qquad \Gamma \vdash t_i \text{ for } i = 1 \dots n}{\Gamma \vdash O(t_1, \dots, t_n)} \tag{3}$$

Example 2.3. The signature of *explicit nondeterminism* has two operations:

$$\mathsf{or} : 2 \qquad\qquad \mathsf{fail} : 0.$$

Some small examples of terms of this signature are

$$\vdash \mathsf{fail} \qquad\qquad x, y, z \vdash \mathsf{or}(x, \mathsf{or}(y, z)) \qquad\qquad x, y, z \vdash \mathsf{or}(\mathsf{or}(x, y), \mathsf{fail}).$$

Example 2.4. The signature of *mutable state* of a single bit has operations:

$$\mathsf{put}^0 : 1 \qquad\qquad \mathsf{put}^1 : 1 \qquad\qquad \mathsf{get} : 2.$$

The informal intuition for a term $\Gamma \vdash \mathsf{put}^i(t)$ is a program that writes the bit $i \in \{0, 1\}$ to the mutable state and then continues as another program t, and a term $\Gamma \vdash \mathsf{get}(t_0, t_1)$ is a program that reads the state, and continues as t_i if the state is i. For example, the term $x, y \vdash \mathsf{put}^0(\mathsf{get}(x, y))$ first writes 0 to the state, then reads 0 from the state, so always continues as x. For simplicity we consider a single bit, but multiple fixed locations and other storage are possible [29].

Definition 2.5. A (first-order finitary) *algebraic theory* $\mathcal{T} = \langle \Sigma, E \rangle$ is a signature Σ (Definition 2.2) and a set E of *equations* of the signature Σ, where an equation is a pair of terms $\Gamma \vdash L$ and $\Gamma \vdash R$ under some context Γ. We will usually write an equation as $\Gamma \vdash L = R$.

Example 2.6. The theory of *exception throwing* has a signature containing a single operation $\mathsf{throw} : 0$ and no equations. The intuition for throw is that it throws an exception and the control flow never comes back, so it is a nullary operation.

Example 2.7. The theory of *explicit nondeterminism* has the signature in Example 2.3 and the following equations saying that fail and or form a monoid:

$$x \vdash \mathsf{or}(\mathsf{fail}, x) = x \quad x \vdash \mathsf{or}(x, \mathsf{fail}) = x \quad x, y, z \vdash \mathsf{or}(x, \mathsf{or}(y, z)) = \mathsf{or}(\mathsf{or}(x, y), z).$$

Following Plotkin and Pretnar [32] we refer to this as "explicit" nondeterminism as it does not include full symmetry laws (that is, $\mathsf{or}(x, y) = \mathsf{or}(y, x)$) or idempotence laws ($\mathsf{or}(x, x) = x$).

Example 2.8. The theory of *mutable state* of a single bit has the signature in Example 2.4 and the following equations for all $i, i' \in \{0, 1\}$:

$$x_0, x_1 \vdash \mathsf{put}^i(\mathsf{get}(x_0, x_1)) = \mathsf{put}^i(x_i) \tag{4}$$

$$x \vdash \mathsf{put}^i(\mathsf{put}^{i'}(x)) = \mathsf{put}^{i'}(x) \tag{5}$$

$$x \vdash \mathsf{get}(\mathsf{put}^0(x), \mathsf{put}^1(x)) = x. \tag{6}$$

The conspicuously missing equation for a get after a get

$$x_{00}, x_{01}, x_{10}, x_{11} \vdash \mathsf{get}(\mathsf{get}(x_{00}, x_{01}), \mathsf{get}(x_{10}, x_{11})) = \mathsf{get}(x_{00}, x_{11})$$

can be derived in the equational logic of algebraic theories, which will be introduced in a more general setting later in Section 3, from the equations above:

$\mathsf{get}(\mathsf{get}(x_{00}, x_{01}), \mathsf{get}(x_{10}, x_{11}))$	via Equation (6), calling this term by t
$= \mathsf{get}(\mathsf{put}^0(t), \mathsf{put}^1(t))$	via Equation (4) for $\mathsf{put}^0(t)$ and $\mathsf{put}^1(t)$
$= \mathsf{get}(\mathsf{put}^0(\mathsf{get}(x_{00}, x_{01})), \mathsf{put}^1(\mathsf{get}(x_{10}, x_{11})))$	via Equation (4) again
$= \mathsf{get}(\mathsf{put}^0(x_{00}), \mathsf{put}^1(x_{11}))$	via Equation (4) right-to-left
$= \mathsf{get}(\mathsf{put}^0(\mathsf{get}(x_{00}, x_{11})), \mathsf{put}^1(\mathsf{get}(x_{00}, x_{11})))$	via Equation (6)
$= \mathsf{get}(x_{00}, x_{11})$	

Remark 2.9. Every algebraic theory gives rise to a monad on the category **Set** of sets by the *free-algebra construction*, which we will discuss in a more general setting in Section 4.2. The three examples above respectively give rise to the monads $(1 + -)$, List, and $(- \times \mathbb{2})^{\mathbb{2}}$ which are used to give semantics to the respective computational effects in programming languages [22, 23].

In this way, the monad for a computational effect is constructed in a remarkably intuitive manner, and this approach is highly composable: One can take the disjoint union of two algebraic theories to combine two effects, and possibly add more equations to characterize the interaction between the two theories [13]. By contrast, coproducts of monads, which correspond to taking the disjoint union of algebraic theories, are much harder to describe explicitly even when they exist.

The kind of plain algebraic theory encapsulated by Definition 2.5 above is not, however, sufficiently expressive for some programming language applications. In this article we focus on two problems with plain algebraic theories:

(1) Firstly, monadic bind for the monad generated by an algebraic theory is defined using *simultaneous substitution of terms*: Given a term $t \in \mathsf{Tm}_\Sigma(\Gamma)$ in a context Γ and a mapping $\sigma : \Gamma \to \mathsf{Tm}_\Sigma(\Gamma')$ from variables in Γ to terms in some context Γ', the monadic bind $t \gg\!= \sigma$ is defined to be the simultaneous substitution $t[\sigma]$ of σ in t:

$$x[\sigma] = \sigma(x) \qquad O(t_1, \ldots, t_n)[\sigma] = O(t_1[\sigma], \ldots, t_n[\sigma]).$$

Monadic bind for a monad is used for interpreting *sequential composition* of computations. Therefore, the second clause above implies that every algebraic effect operation *must* commute

with sequential composition. However, in practice not every effectful operation enjoys this property.

(2) Secondly, it is common to have *multiple instances* of a computational effect that can be dynamically created. For example, it is typical in practice to have an effectful operation openFile that creates a "file descriptor" for a file at a given path, and for each file descriptor there is a pair of read and write operations that are independent of those for other files.

These two restrictions have been studied separately, and different extensions to algebraic theories generalizing Definition 2.5 have been proposed for each: *scoped algebraic effects* for the first problem above and *parameterized algebraic effects* for the second. At first glance, the two problems seem unrelated, but the key insight of this article is that scoped effects can be fruitfully understood as a *non-commutative linear* variant of parameterized effects.

2.2 Scoped Effects

The first problem with plain algebraic theories above is that operations must commute with sequential composition. Therefore an operation $O(a_1, \ldots, a_n)$ is "atomic" in the sense that it may not delimit a fresh *scope*. Alas, in practice it is not uncommon to have operations that do delimit scopes. An example is *exception catching*: $\mathsf{catch}(p, h)$ is a binary operation on computations that first tries the program p and if p throws an exception then h is run. The catch operation does not commute with sequential composition as $\mathsf{catch}(p, h) \gg\!= f$ behaves differently from $\mathsf{catch}(p \gg\!= f, h \gg\!= f)$. The former catches only the exceptions in p whereas the latter catches exceptions both in p and in f. Further examples include nondeterminism with a semi-determinism operator, nondeterminism with cut, and state with local values, which we explore in detail in Section 3.2.

Operations delimiting scopes are treated as *handlers* (i.e., models) of algebraic operations by Plotkin and Pretnar [32], instead of operations in their own right. The following alternative perspective was first advocated by Wu et al. [44].

Perspective 2.10 (Wu et al. [44]). *Since sequential composition of monads generated from algebraic theories is* substitution*, scoped operations are* operations that do not commute with substitution.

Operations that do not commute with substitution arise in contexts other than computational effects as well, for example, in some presentations of the *later modality* of guarded recursion [4].

We can use the results of Plotkin and Power [30] to give a precise phrasing of this on the side of semantics. For simplicity we restrict this definition to the case of monads on **Set**.

Definition 2.11 (Plotkin and Power [30]). A monad T on **Set** can be said to *support an algebraic operation* $O : n$ if it is equipped with a natural transformation

$$\widehat{O}_A : (TA)^n \to TA$$

which moreover satisfies

$$\mu_A \circ \widehat{O}_{TA} = \widehat{O}_A \circ (\mu_A)^n. \tag{7}$$

For example, any free monad T determined by an algebraic theory $\mathcal{T} = \langle \Sigma, E \rangle$ according to Remark 2.9 supports all the operations in the signature Σ. As shown by Plotkin and Power [30], Equation (7) is another phrasing of commuting with sequential composition, and indeed to give such a support for a signature Σ is to simply give a family $(\check{O} \in T(n) \mid (O : n) \in \Sigma)$ of "generic effects," for then we can write

$$\widehat{O}_A(\vec{t}) = \check{O} \gg\!= \vec{t}. \tag{8}$$

Hence we can make a precise definition of scoped operations by dropping (7), following Yang and Wu [46].

Definition 2.12 (Yang and Wu [46]). Let T be a monad on **Set**. A *scoped operation on the monad* T, of arity $k \in \mathbb{N}$, is a family of functions natural in A:

$$\sigma_A : (TA)^k \to TA.$$

(Equation (7) is not necessarily satisfied.)

Example 2.13. Under Perspective 2.10, semi-determinism could be modeled by the monad $\mathsf{List} : \mathbf{Set} \to \mathbf{Set}$ for explicit nondeterminism equipped with a scoped operation once of arity 1, given by the family of functions $\mathsf{once}_A : \mathsf{List}(A) \to \mathsf{List}(A)$ where

$$\mathsf{once}_A([]) = [], \qquad \mathsf{once}_A([x, \ldots]) = [x].$$

This family of functions is natural in A but does not satisfy (7).

Extensions of effect handlers to natively accommodate scoped operations were first studied by Wu et al. [44] in Haskell, where the authors proposed two approaches:

(1) The *bracketing approach* uses a pair of *algebraic* operations begin_s and end_s to encode a scoped operation s. For example, consider the program $s(\mathsf{put}^0); \mathsf{put}^1; x$, expressed in direct style, which writes the value 0 in the scope s, then writes the value 1 outside of that scope and continues with the rest of the program x. This can be encoded formally as

$$\mathsf{begin}_s(\mathsf{put}^0(\mathsf{end}_s(\mathsf{put}^1(x)))).$$

(2) The *higher-order syntax approach* directly constructs the syntax for programs with algebraic and scoped operations as an initial algebra of an endofunctor over the category of (finitary) endofunctors over **Set**. Concretely, this amounts to adding the following rule, for every scoped operation S that delimits n scopes, to the rules in (3) generating terms:

$$\frac{S : n \qquad x_1, \ldots, x_m \vdash t_i \text{ for } i = 1 \ldots n \qquad \Gamma \vdash k_j \text{ for } j = 1 \ldots m}{\Gamma \vdash S(t_1, \ldots, t_n)\{k_1, \ldots, k_m\}}.$$

The intuition is that $S(t_1, \ldots, t_n)\{k_1, \ldots, k_m\}$ is the term applying the scoped operation S to the terms $t_1, \ldots, t_n$ followed by a "substitution" that replaces the m variables in t_i with the terms $k_1, \ldots, k_m$ respectively (c.f. explicit substitution [11] and delayed substitution [4]). Since substitutions get stuck at scoped operations, they need to be kept in terms, if we want to have a monad of terms with scoped operations. However, we emphasize that the "substitution" $\{k_1, \ldots, k_m\}$ is only an informal intuition (so "substitution" is written in quotation); formally, it is just a part of the term $S(t_1, \ldots, t_n)\{k_1, \ldots, k_m\}$ rather than applying the (real) substitution $[k_1, \ldots, k_m]$ operation to the term $S(t_1, \ldots, t_n)$.

Moreover, to make the "substitution" $\{k_1, \ldots, k_m\}$ behave like real substitutions, the terms are further quotiented by (the congruence relation generated by) the following rule: for all $m, m' \in \mathbb{N}$ and functions $f : \{1 \ldots m\} \to \{1 \ldots m'\}$,

$$\frac{S : n \qquad x_1, \ldots, x_m \vdash t_i \text{ for } i = 1 \ldots n \qquad \Gamma \vdash k_{j'} \text{ for } j' = 1 \ldots m'}{\Gamma \vdash S(t_1, \ldots, t_n)\{k_{f(1)}, \ldots, k_{f(m)}\} = S(t_1[f], \ldots, t_n[f])\{k_1, \ldots, k_{m'}\}},$$

where the terms $x_1 \ldots x_{m'} \vdash t_i[f]$ are obtained by replacing each variable x_j in t_i with $x_{f(j)}$, for $1 \le j \le m$. This rule is motivated as follows: In the left-hand side, every variable x_j in each t_i is "replaced" by $k_{f(j)}$ according to the "substitution" $\{k_{f(1)}, \ldots, k_{f(m)}\}$, while in the right-hand side, x_j is first replaced by $x_{f(j)}$ by the real substitution $[f]$, and then "replaced"

by $k_{f(j)}$ according to the "substitution" $\{k_1, \ldots, k_{m'}\}$. These two results should be exactly the same since in the end both replace every x_j with $k_{f(j)}$.

Terms in a context Γ obtained in this way also form a monad [45].

The higher-order syntax approach was regarded the more principled one since in the first approach ill bracketed pairs of begin_s and end_s are possible, such as

$$\mathsf{end}_s(\mathsf{put}^0(\mathsf{begin}_s(\mathsf{begin}_s(\mathsf{put}^1(x))))).$$

In subsequent work, both of these two approaches received further development [25, 43, 45, 46] and operational semantics for scoped effects has also been developed [5]. Of particular relevance to the current article is the work of Piróg et al. [25], which we briefly review in the rest of this section.

Related Work on Models for Scoped Effects. Piróg et al. [25] fix the ill-bracketing problem in the bracketing approach by considering the category $\mathbf{Set}^{\mathbb{N}}$ whose objects are sequences $X = (X(0), X(1), \ldots)$ of sets and morphisms are just sequences of functions. (In other words, this is the category of functors $\mathbb{N} \to \mathbf{Set}$ and natural transformations between them, where $\mathbb{N}$ is the discrete category whose objects are natural numbers and where all morphisms are identity morphisms.) Given $X \in \mathbf{Set}^{\mathbb{N}}$, the idea is that $X(n)$ represents a set of terms at *bracketing level n* for every $n \in \mathbb{N}$.

On this category, there are two functors $(\triangleright), (\triangleleft) : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set}^{\mathbb{N}}$, pronounced "later" and "earlier," that shift the bracketing levels:

$$(\triangleright X)(0) = \emptyset, \qquad (\triangleright X)(n+1) = X(n), \qquad (\triangleleft X)(n) = X(n+1). \tag{9}$$

These two functors are closely related to the bracketing approach (1): a morphism $b : \triangleleft X \to X$ for a functor X opens a scope, turning a term t at level $n+1$ to the term $\mathsf{begin}(t)$ at level n. Conversely, a morphism $e : \triangleright X \to X$ closes a scope, turning a term t outside the scope, so at level $n-1$, to the term $\mathsf{end}(t)$ at level n.

Every signature Σ as in Definition 2.2 determines a functor $\bar{\Sigma} : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set}^{\mathbb{N}}$ given by

$$(\bar{\Sigma}X)(n) = \coprod_{o \in |\Sigma|} X(n)^{ar(o)}. \tag{10}$$

Given two signatures Σ and Σ' (for algebraic and scoped operations respectively), we use (10) with each to obtain two functors $\bar{\Sigma}, \bar{\Sigma}' : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set}^{\mathbb{N}}$. Moreover, for every $A \in \mathbf{Set}$, let $\uparrow A \in \mathbf{Set}^{\mathbb{N}}$ be given by

$$(\uparrow A)(0) = A \qquad (\uparrow A)(n+1) = 0, \tag{11}$$

and conversely for every $X \in \mathbf{Set}^{\mathbb{N}}$, let $\downarrow X \in \mathbf{Set}$ be given by

$$\downarrow X = X(0). \tag{12}$$

These are actually adjoint functors

$$\uparrow : \mathbf{Set} \to \mathbf{Set}^{\mathbb{N}} \qquad \downarrow : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set} \qquad \uparrow \dashv \downarrow. \tag{13}$$

Now Proposition 2.14 constructs the syntactic monad for programs with the given algebraic and scoped operations, without taking into account any equations.

Proposition 2.14 (Piróg et al. [25]). *The following functor can be extended to a monad that is isomorphic to the monad obtained by the higher-order syntax approach (2):*

$$\downarrow \circ \left(\bar{\Sigma} + \left(\bar{\Sigma}' \circ \triangleleft\right) + \triangleright\right)^* \circ \uparrow : \mathbf{Set} \to \mathbf{Set},$$

where $(-)^*$ is the free monad over an endofunctor.

Piróg et al. [25] define a *model* of a scoped effect as an *algebra* for the monad $(\bar{\Sigma} + (\bar{\Sigma}' \circ \triangleleft) + \triangleright)^*$ on $\mathbf{Set}^{\mathbb{N}}$. This is a notion of handler accommodating both algebraic and scoped effects. In Theorem 4.10 we show that this monad on $\mathbf{Set}^{\mathbb{N}}$ arises as a special case of the free monad for a parameterized algebraic theory. In Theorems 4.12–4.14, we show that three examples of models of Piróg et al. [25] are actually *free algebras* on $\uparrow A \in \mathbf{Set}^{\mathbb{N}}$ for an appropriate set of equations for each example.

2.3 Parameterized Algebraic Theories

Recall that our second problem with plain algebraic theories is that they do not support the dynamic creation of multiple instances of computational effects. This problem, sometimes known as the *local computational effects* problem, was first systematically studied by Power [35] in a purely categorical setting. A syntactic framework extending that of algebraic theories, called *parameterized algebraic theories*, was introduced by Staton [37, 38] and is used to give an axiomatic account of local computational effects such as restriction [27], local state [29], and the π-calculus [21, 36].

Operations in a parameterized theory are more general than those in an algebraic theory because they may *use* and *create* values in an *abstract* type of parameters. The parameter type has different intended meanings for different examples of parameterized theories, typically as some kind of resource such as memory locations or communication channels. In this article, we propose to interpret parameters as *names of scopes*.

Perspective 2.15. *Scoped operations can be understood as operations allocating and consuming instances of a resource: the names of scopes.*

In the case of local state, the operations of Example 2.4 become $\mathsf{get}(a; x_0, x_1)$ and $\mathsf{put}^i(a; x)$, now taking a parameter a which is the location being read or written to. In a sense, each memory location a represents an *instance* of the state effect, with its own get and put operations. We also have a term $\mathsf{new}^i(a.x(a))$ which allocates a fresh location named a storing an initial value i, then continues as x; the computation x might mention location a. The following is a possible equation, which says that reading immediately after allocating is redundant:

$$\mathsf{new}^i(a.\mathsf{get}(a, x_0(a), x_1(a))) = \mathsf{new}^i(a.x_i(a)).$$

For the full axiomatization of local state see [38, Section V.E]. A closed term can only mention locations introduced by new^i, meaning that the type of locations is *abstract*.

To model scoped operations, we think of them as allocating a new scope. For example, the scoped operation once, which chooses the first non-failing branch of a nondeterministic computation, is written as $\mathsf{once}(a.x(a))$. It creates a new scope a and proceeds as x. As in Section 1, there is an explicit operation $\mathsf{close}(a; x)$ for closing the scope a and continuing as x.

Well-formed programs close scopes precisely once and in the reverse order to their allocation. Thus in Section 3 we will discuss a *non-commutative linear* variation of parameterized algebraic theories needed to model scoped effects. With our framework we then give axiomatizations for examples from the scoped effects literature (Theorems 4.12–4.14, 4.16).

Our parameters are linear in the same sense as variables in linear logic and linear lambda calculi [3, 12], but with an additional non-commutativity restriction. Non-commutative linear systems are also known as ordered linear systems [24, 33]. A commutative linear version of parameterized algebraic theories was considered by Staton [40] to give an algebraic theory of quantum computation; in this case, parameters stand for qubits.

Remark 2.16. Parameterized algebraic theories correspond to a certain class of enriched monads [37], extending the correspondence between algebraic theories and monads on the category of sets, and the idea of Plotkin and Power [29] that computational effects give rise to monads (see Section 2.1). Thus, the syntactic framework of parameterized theories has a canonical semantic status.

We can use the monad arising from a parameterized theory to give semantics to a programming language containing the effects in question.

The framework of parameterized algebraic theories is related to graded theories [14], which also use presheaf-enrichment; second-order algebra [7–9], which also uses variable binding; and graphical methods [19], which also connect to presheaf categories.

3 Parameterized Theories of Scoped Effects

In this section we describe scoped effects (Section 2.2) in terms of parameterized algebraic theories (Section 2.3). To do this, we use a substructural version of parameterized algebraic theories. The parameters will be used to interpret the names of scopes, and the substructural treatment mirrors the idea that scopes cannot be implicitly duplicated, reordered, or discarded.

A parameterized algebraic theory consists of a signature (Definition 3.1) and equations (Definition 3.4) between terms formed from the signature. We show that every scoped signature gives rise to a parameterized algebraic *signature* (Section 3.1, Example 3.3). But a parameterized algebraic *theory* includes equations as well as operations. In Section 3.2 we propose equational theories for various scoped effects: exceptions, state, and nondeterminism.

This section is focused on syntactic aspects of parameterized algebraic theories. In Section 4, we turn to models of the theories. In Section 4.4, we show that the free models of the theories of exceptions, state, and nondeterminism completely capture earlier notions from the literature.

3.1 Parameterized Algebraic Theories with Non-Commutative Linear Parameters

Definition 3.1. A *(parameterized) signature* $\Sigma = \langle |\Sigma|, ar \rangle$ consists of a set of operations $|\Sigma|$ and for each operation $\mathrm{O} \in |\Sigma|$ a *parameterized arity* $ar(\mathrm{O}) = (p \mid m_1, \ldots, m_k)$ consisting of a natural number p and a list of natural number *valences* $m_1, \ldots, m_k$. This means that the operation O takes in p parameters and k continuations, and it binds m_i parameters in the ith continuation.

Example 3.2. The algebraic theory of explicit nondeterminism in Example 2.3 can be extended with a *semi-determinism* operator once:

$$\mathsf{or} : (0 \mid 0, 0) \qquad \mathsf{once} : (0 \mid 1) \qquad \mathsf{fail} : (0 \mid -) \qquad \mathsf{close} : (1 \mid 0).$$

(We write $-$ for an empty list or context.) The arity of or says that or takes no parameters and has two continuations, each binding no parameters. The operation once also takes no parameters and has one continuation that opens a new scope by binding a parameter. Inside this scope, only the first successful branch of or is kept. The operation fail takes zero parameters and no continuations; close takes one parameter and has one continuation that binds no parameter. We consider the terms and equations in Example 3.5 and Figure 4.

Example 3.3. Given signatures for algebraic and scoped operations, as in Definition 2.2 and Section 2.2, we can translate them to a parameterized signature as follows:

- for each algebraic operation $(\mathsf{op} : k)$ of arity $k \in \mathbb{N}$, there is a parameterized operation with arity $(0 \mid 0, \ldots, 0)$, where the list $0, \ldots, 0$ has length k;
- for each scoped operation $(\mathsf{sc} : k)$ of arity $k \in \mathbb{N}$, there is a parameterized operation $\mathsf{sc} : (0 \mid 1, \ldots, 1)$, where the list $1, \ldots, 1$ has length k;
- there is an operation $\mathsf{close} : (1 \mid 0)$, which closes the most recent scope, and which all the different scoped operations share.

Not every parameterized signature arises from a scoped signature, because in general we may have operations where the arguments have different valences (e.g., $m_1 \neq m_2$). So parameterized signatures give some more flexibility. We explore this point more in Example 3.7.

For a given signature, we define the terms-in-context of *algebraic theories with non-commutative linear parameters*. Terms contain two kinds of variables: computation variables (x, y, ...), which each expect a certain number of parameters, and parameter variables (a, b, ...). In the case of scoped effects, a parameter represents the name of a scope.

A context Γ of *computation variables* is a finite list $x_1 : p_1, \dots, x_n : p_n$, where each variable x_i is annotated with the number p_i of parameters it consumes. A context Δ of *parameter variables* is a finite list $a_1, \dots, a_m$. Terms $\Gamma \mid \Delta \vdash t$ are inductively generated as follows:

$$\frac{}{\Gamma, x : p, \Gamma' \mid a_1, \dots, a_p \vdash x(a_1, \dots, a_p)} \tag{14}$$

$$\frac{\Gamma \mid \Delta, b_1, \dots, b_{m_1} \vdash t_1 \qquad \dots \qquad \Gamma \mid \Delta, b_1, \dots, b_{m_k} \vdash t_k \qquad \mathrm{O} : (p \mid m_1, \dots, m_k)}{\Gamma \mid \Delta, a_1, \dots, a_p \vdash \mathrm{O}(a_1, \dots, a_p;\ b_1 \dots b_{m_1}.t_1,\ \dots,\ b_1 \dots b_{m_k}.t_k)}. \tag{15}$$

In the conclusion of the last rule, the parameters $a_1, \dots, a_p$ are consumed by the operation O. The notation $b_1 \dots b_{m_i}.t_i$ indicates that the parameters $b_1, \dots, b_{m_i}$ are bound in t_i and is kept simpler by writing the list of bound parameters without any delimiters. As usual, we treat all terms up to renaming of variables.

We make use of several conveniences when writing terms.

—In (14), if $p = 0$ we write "$x(-)$" simply as "x."
—In (15), if $m_i = 0$ we write "$-.t_i$" simply as "t_i."
—In (15), if $p = 0 = k$, we write "$\mathrm{O}(-;-)$" as O, or
—if only $p = 0$ we omit "$-;$" (and if only $k = 0$ we omit "$;-$").

Before expanding on the meaning of terms $\Gamma \mid \Delta \vdash t$ in general, we note how the terms of an ordinary algebraic theory described in (3) can be interpreted as terms with parameters. We first define the interpretation $(-)^0$ on contexts into computation contexts:

$$(-)^0 = -$$
$$(\Gamma, x)^0 = \Gamma^0, x : 0$$

by considering each variable of ordinary algebra to be a computation variable that consumes zero parameters. Then on term judgements we have an interpretation

$$(\Gamma \vdash t)^0 = \Gamma^0 \mid - \vdash t^0,$$

where our shorthand makes t^0 identical to t, but for clarity if we used the full notation we would have the following:

$$x^0 = x(-)$$
$$(\mathsf{op}(t_1, \dots, t_k))^0 = \mathsf{op}(-; -.t_1^0, \dots, -.t_k^0).$$

The context Γ of computation variables admits the usual structural rules: weakening, contraction, and exchange; the context Δ of parameters does not. All parameters in Δ must be used exactly once, in the reverse of the order in which they appear. Intuitively, a parameter in Δ is the name of an open scope, so the restrictions on Δ mean that scopes must be closed eventually and in the opposite order that they were opened, that is, scopes are well-bracketed. We chose the restrictions on Δ to model scoped effects in particular, but these restrictions would not be appropriate for modeling other examples such as file handlers, where files may be closed in any order.

The arguments $t_1, \dots, t_k$ of an operation O are continuations, each corresponding to a different branch of the computation, hence they share the parameter context Δ. Examples where the

$$\frac{\begin{array}{c} x_1 : m_1, \ldots, x_l : m_l \mid a_1, \ldots, a_{m_i} \vdash x_i(a_1, \ldots, a_{m_i}) \\ \Gamma' \mid \Delta', c_1, \ldots, c_{m_1} \vdash t_1 \qquad \ldots \qquad \Gamma' \mid \Delta', c_1, \ldots, c_{m_l} \vdash t_l \end{array}}{\Gamma' \mid \Delta', a_1, \ldots, a_{m_i} \vdash x_i(a_1, \ldots, a_{m_i})\big[t_1/x_1, \ldots, t_l/x_l\big] \coloneqq t_i[a_1/c_1, \ldots, a_{m_i}/c_{m_i}]}$$

$$\frac{\begin{array}{c} x_1 : m_1, \ldots, x_l : m_l \mid \Delta, a_1, \ldots, a_p \vdash \mathrm{O}(a_1, \ldots, a_p;\ b_1 \ldots b_{m'_1}.s_1, \ldots, b_1 \ldots b_{m'_k}.s_k) \\ \Gamma' \mid \Delta', c_1, \ldots, c_{m_1} \vdash t_1 \qquad \ldots \qquad \Gamma' \mid \Delta', c_1, \ldots, c_{m_l} \vdash t_l \end{array}}{\begin{array}{l} \Gamma' \mid \Delta', \Delta, a_1, \ldots, a_p \vdash \mathrm{O}(a_1, \ldots, a_p;\ b_1 \ldots b_{m'_1}.s_1, \ldots, b_1 \ldots b_{m'_k}.s_k)\big[t_1/x_1, \ldots, t_l/x_l\big] \\ \qquad \coloneqq \\ \qquad \mathrm{O}(a_1, \ldots, a_p;\ b_1 \ldots b_{m'_1}.s_1\big[t_1/x_1, \ldots, t_l/x_l\big], \ldots, b_1 \ldots b_{m'_k}.s_k\big[t_1/x_1, \ldots, t_l/x_l\big]) \end{array}}$$

Fig. 2. Action of substitution on terms. In the first rule, $[a_1/c_1, \ldots, a_{m_i}/c_{m_i}]$ represents the obvious renaming of free parameters in t_i.

continuations $t_1, \ldots, t_k$ run in parallel, jointly consuming Δ, cannot be expressed in our flavor of parameterized theory.

Compared to the *algebra with linear parameters* used for describing quantum computation [40], our syntactic framework has the additional constraint that Δ cannot be reordered. Given these constraints, the context Δ is in fact a stack, so inside a term it is unnecessary to refer to the variables in Δ by name. We have chosen to do so anyway in order to make clearer the connection to non-linear parameterized theories [37, 38].

The syntax admits the following simultaneous substitution rule:

$$\frac{x_1 : m_1, \ldots, x_l : m_l \mid \Delta \vdash t \qquad \Gamma' \mid \Delta', a_1, \ldots, a_{m_1} \vdash t_1 \qquad \ldots \qquad \Gamma' \mid \Delta', a_1, \ldots, a_{m_l} \vdash t_l}{\Gamma' \mid \Delta', \Delta \vdash t\big[(\Delta', a_1, \ldots, a_{m_1} \vdash t_1)/x_1 \ \ldots \ (\Delta', a_1, \ldots, a_{m_l} \vdash t_l)/x_l\big]}. \tag{16}$$

In the conclusion, the notation $(\Delta', a_1, \ldots, a_{m_i} \vdash t_i)/x_i$ emphasizes that the parameters $a_1, \ldots, a_{m_i}$ in t_i are replaced by the corresponding parameters that x_i consumes in t, either bound parameters or free parameters from Δ. To ensure that the term in the conclusion is well-formed, we must substitute a term that depends on Δ' for *all* the computation variables in the context of t. The action of (well-typed) substitution on terms is defined inductively by the rules in Figure 2.

An important special case of the substitution rule is where we add a number of extra parameter variables to the beginning of the parameter context, increasing the sort of each computation variable by the same number. The following example instance of (16), where $ar(\mathrm{O}) = (1 \mid 1)$, illustrates such a "weakening" by adding two extra parameter variables a'_1, a'_2 and replacing $x : 2$ by $x' : 4$:

$$\frac{x : 2 \mid a_1, a_2 \vdash \mathrm{O}(a_2;\ b.x(a_1, b)) \qquad x' : 4 \mid a'_1, a'_2, b_1, b_2 \vdash x'(a'_1, a'_2, b_1, b_2)}{x' : 4 \mid a'_1, a'_2, a_1, a_2 \vdash \mathrm{O}(a_2;\ b.x'(a'_1, a'_2, a_1, b))}.$$

Definition 3.4. An *algebraic theory* $\mathcal{T} = \langle \Sigma, E \rangle$ *with non-commutative linear parameters* is a parameterized signature Σ together with a set E of *equations*. An equation is a pair of terms $\Gamma \mid \Delta \vdash L$ and $\Gamma \mid \Delta \vdash R$ in the same context $(\Gamma \mid \Delta)$. We write an equation as $\Gamma \mid \Delta \vdash L = R$.

We will omit the qualifier "with non-commutative linear parameters" where convenient and refer to "parameterized theories" or just "theories."

Given a theory $\mathcal{T} = \langle \Sigma, E \rangle$, we form an equivalence relation $=_{\mathcal{T}}$ on terms of $\mathcal{T}$, called the *derivable equality*, by closing *substitution instances* of the equations in E under reflexivity, symmetry, transitivity, and congruence rules (Figure 3).

$$\frac{(x_1 : m_1, \ldots, x_l : m_l \mid \Delta \vdash L =_{\mathcal{T}} R) \in E \qquad \Gamma' \mid \Delta', a_1, \ldots, a_{m_1} \vdash t_1 \quad \ldots \quad \Gamma' \mid \Delta', a_1, \ldots, a_{m_l} \vdash t_l}{\Gamma' \mid \Delta', \Delta \vdash L\big[t_1/x_1, \ldots, t_l/x_l\big] =_{\mathcal{T}} R\big[t_1/x_1, \ldots, t_l/x_l\big]}$$

$$\frac{\Gamma \mid \Delta \vdash t}{\Gamma \mid \Delta \vdash t =_{\mathcal{T}} t} \qquad \frac{\Gamma \mid \Delta \vdash L =_{\mathcal{T}} R}{\Gamma \mid \Delta \vdash R =_{\mathcal{T}} L} \qquad \frac{\Gamma \mid \Delta \vdash L =_{\mathcal{T}} R \qquad \Gamma \mid \Delta \vdash R =_{\mathcal{T}} R'}{\Gamma \mid \Delta \vdash L =_{\mathcal{T}} R'}$$

$$\frac{\Gamma \mid \Delta, b_1, \ldots, b_{m_1} \vdash L_1 =_{\mathcal{T}} R_1 \qquad \ldots \qquad \Gamma \mid \Delta, b_1, \ldots, b_{m_k} \vdash L_k =_{\mathcal{T}} R_k \qquad \mathrm{O} : (p \mid m_1, \ldots, m_k) \in \Sigma}{\begin{array}{rl} & \mathrm{O}(a_1, \ldots, a_p;\ b_1 \ldots b_{m_1}.L_1, \ldots, b_1 \ldots b_{m_k}.L_k) \\ \Gamma \mid \Delta, a_1, \ldots, a_p \vdash & =_{\mathcal{T}} \\ & \mathrm{O}(a_1, \ldots, a_p;\ b_1 \ldots b_{m_1}.R_1, \ldots, b_1 \ldots b_{m_k}.R_k) \end{array}}$$

Fig. 3. The rules for the derivable equality on terms of a parameterized algebraic theory.

$$x : 0, y : 0, z : 0 \mid - \vdash \mathsf{or}(\mathsf{or}(x, y), z) = \mathsf{or}(x, \mathsf{or}(y, z)) \tag{17}$$
$$x : 0 \mid - \vdash \mathsf{or}(x, \mathsf{fail}) = x \tag{18}$$
$$x : 0 \mid - \vdash \mathsf{or}(\mathsf{fail}, x) = x \tag{19}$$
$$- \mid - \vdash \mathsf{once}(a.\mathsf{fail}) = \mathsf{fail} \tag{20}$$
$$x : 1 \mid - \vdash \mathsf{once}(a.\mathsf{or}(x(a),\ x(a))) = \mathsf{once}(a.x(a)) \tag{21}$$
$$x : 0 \mid - \vdash \mathsf{once}(a.\mathsf{close}(a; x)) = x \tag{22}$$
$$x : 0, y : 1 \mid - \vdash \mathsf{once}\big(a.\mathsf{or}(\mathsf{close}(a; x),\ y(a))\big) = x \tag{23}$$

Fig. 4. The parameterized theory of explicit nondeterminism (17)–(19) and once (20)–(23).

3.2 Examples of Equations for Scoped Theories via Parameterized Presentations

Example 3.5. We continue Example 3.2, semi-determinism, with or, once, fail, and close. The term formation rules in Section 3.1 allow the most recently opened scope to be closed using the close operation by consuming the most recently bound parameter; close has one continuation which does not depend on any parameters.

Examples of equations, i.e., pairs of terms in the same context, are given in Figure 4. As an illustration, we note that Equation (22) is derivable from the others:

$$\begin{aligned} \mathsf{once}(a.\mathsf{close}(a; x)) &= \mathsf{once}(a.\mathsf{or}(\mathsf{close}(a; x), \mathsf{fail})) && \text{via (18)} \\ &= x && \text{via (23).} \end{aligned}$$

Example 3.6. As we mentioned earlier, *exception catching* is not an ordinary algebraic operation. The signature for throwing and catching exceptions is the following:

$$\mathsf{throw} : (0 \mid -) \qquad \mathsf{catch} : (0 \mid 1, 1) \qquad \mathsf{close} : (1 \mid 0).$$

The throw operation uses no parameters and takes no continuations. The catch operation uses no parameters and takes two continuations which each open a new scope, by binding a fresh parameter. Exceptions are caught in the first continuation and are handled using the second continuation.

The close operation uses one parameter and takes one continuation binding no parameters. The term $\mathsf{close}(a; x)$ closes the scope named by a and continues as x. For example, in the term $\mathsf{catch}(a.\mathsf{close}(a; x), b.y(b))$, exceptions in x will not be caught, because the scope of the catch

operation has already been closed. The equations are

$$y : 0 \mid - \vdash \mathsf{catch}(a.\mathsf{throw},\ b.\mathsf{close}(b; y)) = y \tag{24}$$

$$- \mid - \vdash \mathsf{catch}(a.\mathsf{throw},\ b.\mathsf{throw}) = \mathsf{throw} \tag{25}$$

$$x : 0, y : 1 \mid - \vdash \mathsf{catch}\big(a.\mathsf{close}(a; x),\ b.y(b)\big) = x. \tag{26}$$

Example 3.7. The arity of catch from Example 3.6 corresponds to the signature used in [25, Example 4.5], in which both arguments of catch delimit a scope. In the standard operational behavior of exception catching $\mathsf{catch}(a.p, b.h)$, if an exception is thrown within the scope marked by a, the control flow transfers to h, and when h closes the scope marked by b by some $\mathsf{close}(b; y)$, the program simply continues as y. Therefore the scope b is in fact unimportant, and it seems more natural to model exception catching using the arity $\mathsf{varcatch} : (0 \mid 1, 0)$, where the second argument does not create a new scope. Equations (24)–(26) become then

$$y : 0 \mid - \vdash \mathsf{varcatch}(a.\mathsf{throw},\ y) = y$$

$$x : 0, y : 0 \mid - \vdash \mathsf{varcatch}\big(a.\mathsf{close}(a, x),\ y\big) = x.$$

We could also encode varcatch in the theory of catch by making the following definition:

$$\mathsf{varcatch}(a.x(a), y) = \mathsf{catch}(a.x(a), b.\mathsf{close}(b; y))$$

and deducing the two equations for varcatch from Equations (24) and (26).

Example 3.8 (Mutable State with Local Values). The theory of (Boolean) mutable state with one memory location (Example 2.4) can be extended with scoped operations local^0 and local^1 that write respectively 0 and 1 to the state. Inside the scope of local, the value of the state just before the local is not accessible anymore, but when the local is closed the state reverts to this previous value:

$$\mathsf{local}^i : (0 \mid 1) \qquad \mathsf{put}^i : (0 \mid 0) \qquad \mathsf{get} : (0 \mid 0, 0) \qquad \mathsf{close} : (1 \mid 0).$$

The equations for the parameterized theory of state with local comprise the usual equations for state in the literature [20, 29]:

$$z : 0 \mid - \vdash \mathsf{get}(\mathsf{put}^0(z), \mathsf{put}^1(z)) = z \tag{27}$$

$$z : 0 \mid - \vdash \mathsf{put}^i(\mathsf{put}^j(z)) = \mathsf{put}^j(z) \tag{28}$$

$$x_0 : 0, x_1 : 0 \mid - \vdash \mathsf{put}^i(\mathsf{get}(x_0, x_1)) = \mathsf{put}^i(x_i) \tag{29}$$

together with equations for local/close, and the interaction with state:

$$x : 0 \mid - \vdash \mathsf{local}^i(a.\mathsf{close}(a; x)) = x \tag{30}$$

$$x_0 : 1, x_1 : 1 \mid - \vdash \mathsf{local}^i(a.\mathsf{get}(x_0(a),\ x_1(a))) = \mathsf{local}^i(a.x_i(a)) \tag{31}$$

$$z : 1 \mid - \vdash \mathsf{local}^i(a.\mathsf{put}^j(z(a))) = \mathsf{local}^j(a.z(a)) \tag{32}$$

$$z : 0 \mid a \vdash \mathsf{put}^i(\mathsf{close}(a; z)) = \mathsf{close}(a; z). \tag{33}$$

This extension of mutable state is different from the one discussed in Section 2.3, where memory locations can be dynamically created.

Example 3.9 (Explicit Nondeterminism with Cut). The theory of explicit nondeterminism given by Equations (17) to (19) can be extended with an operation that prunes the list of possible results, similar to cut in Prolog:

$$\mathsf{cut} : (0 \mid 0) \qquad \mathsf{or} : (0 \mid 0, 0) \qquad \mathsf{fail} : (0 \mid -).$$

The cut operation is algebraic and has one continuation; $\mathsf{cut}(x)$ intuitively discards the choices that have not been explored yet, and returns all the possible results of x. The behaviour of cut has been axiomatized in [26, Section 6]:

$$x : 0, y : 0 \mid - \vdash \mathsf{or}(\mathsf{cut}(x), y) = \mathsf{cut}(x) \tag{34}$$

$$x : 0, y : 0 \mid - \vdash \mathsf{or}(x, \mathsf{cut}(y)) = \mathsf{cut}(\mathsf{or}(x, y)) \tag{35}$$

$$x : 0 \mid - \vdash \mathsf{cut}(\mathsf{cut}(x)) = \mathsf{cut}(x). \tag{36}$$

To delimit the scope in which cut discards choices, we add an operation that opens a scope, and a close operation for closing scopes:

$$\mathsf{scope} : (0 \mid 1) \qquad\qquad \mathsf{close} : (1 \mid 0).$$

We propose the following equations for scope:

$$- \mid - \vdash \mathsf{scope}(a.\mathsf{fail}) = \mathsf{fail} \tag{37}$$

$$x : 0 \mid - \vdash \mathsf{scope}(a.\mathsf{cut}(x(a))) = \mathsf{scope}(a.x(a)) \tag{38}$$

$$x : 0, y : 1 \mid - \vdash \mathsf{scope}(a.\mathsf{or}(\mathsf{close}(a; x),\ y(a))) = \mathsf{or}(x,\ \mathsf{scope}(a.y(a))). \tag{39}$$

Intuitively, Equation (38) says that when a cut reaches the boundary of a scope, the cut is erased so it cannot affect choices outside of the scope. The Haskell implementation of the `scope` function by Piróg and Staton [26] has similar behavior.

Equation (39) axiomatizes the interaction between opening and closing a scope. From it, we can derive the following:

$$\begin{aligned}
\mathsf{scope}(a.\mathsf{close}(a; x)) &= \mathsf{scope}(a.\mathsf{or}(\mathsf{close}(a; x), \mathsf{fail})) && \text{via (18)}\\
&= \mathsf{or}(x, \mathsf{scope}(a.\mathsf{fail})) && \text{via (39)}\\
&= \mathsf{or}(x, \mathsf{fail}) && \text{via (37)}\\
&= x && \text{via (18).}
\end{aligned}$$

Remark 3.10. We can almost encode the theory of once from Example 3.5 into the theory for cut and scope (Example 3.9), by defining once to be scope, and defining the $\mathsf{close}(a; x)$ of the once theory to be $\mathsf{cut}(\mathsf{close}(a; x))$. Then we can recover Equations (20), (22), and (23) from Figure 4, by using Equations (34) to (39), but we cannot recover Equation (21), which is idempotence of or inside a once.

4 Models of Parameterized Theories

In the previous section, we introduced parameterized algebraic theories, showing how scoped signatures induce parameterized signatures (Example 3.3), and giving four examples of parameterized theories with equations in the scoped setting: nondeterminism with once, catching exceptions, state, and nondeterminism with cut (Examples 3.5–3.9). The previous section was concerned with syntactic aspects of parameterized theories. In this section, we look at semantic aspects, via models.

The main results of this section are in Section 4.4, where we connect the free models of the four examples of parameterized theories with constructions from the literature.

To formulate these main results, we first look at models of parameterized algebraic theories generally, in Section 4.1. We show that an appropriate setting for models is the category $\mathbf{Set}^{\mathbb{N}}$, where the objects are sequences of sets and the morphisms are sequences of functions. The rough idea is that a model has a carrier for every possible parameter context, i.e., every possible scoping depth. Models of parameterized algebraic theories have been studied before, but here we use a substructural variation which has a different notion of model. We then formulate and characterize

free models in Section 4.2. We show that free models form a strong monad, by analogy with the first order case.

Although the free models of parameterized algebraic theories naturally live in the category $\mathbf{Set}^{\mathbb{N}}$, under Perspective 2.10 scoped effects are operations on a monad on **Set**, i.e., without the indexed family of sets. Moreover, in some previous work on handling scoped effects, the programs of the domain-specific language being handled form a monad on **Set**. We make the connection to both of these in Section 4.3, observing that a parameterized algebraic theory also gives rise to a monad on **Set** (with scoped operations), and in particular we show that a previously considered monad for scoped effects on **Set** arises in this way (Theorem 4.10).

We connect the free models of the four examples of parameterized theories with constructions from the literature in Section 4.4. Finally, in Section 4.5, we use the model-based foundation to construct parameterized algebraic theories for scoped effects in general, beyond the four examples of Section 3.2. This shows that our new Perspective 2.15 allows for finer distinctions than Perspective 2.10, since there are multiple possible parameterized algebraic theories consistent with a given monad and scoped operations, a fact we demonstrate for the List monad and once in Proposition 4.19.

4.1 Models in $\mathbf{Set}^{\mathbb{N}}$

A model of a first-order algebraic theory (e.g., [2]) consists simply of a set together with specified interpretations of the operations of the signature, validating a (possibly empty) equational specification. The more complex arities and judgement forms of a parameterized theory require a correspondingly more complex notion of model. Rather than simply being a set of abstract computations, a model will now be stratified into a sequence of sets $X = (X(0), X(1), \ldots) \in \mathbf{Set}^{\mathbb{N}}$ where $X(n)$ represents computations *taking n parameters*. In Section 2.2 we described the somewhat different use of $\mathbf{Set}^{\mathbb{N}}$ by Piróg et al's [25]. We connect the two approaches in Theorem 4.10 below.

Intuitively, an object n in the discrete category $\mathbb{N}$ can be thought of as a parameter context Δ with n parameters. As explained in Section 3.1, parameter contexts do not admit weakening, contraction, or exchange, making the use of a discrete category appropriate for representing parameter contexts. Work closely related to ours employs other categories instead of $\mathbb{N}$: In the algebra with linear parameters of [40] only parameter exchange is allowed, so the category used is Bij, consisting of natural numbers, regarded as finite sets, and bijections between them. The parameterized theories of [38] allow all structural rules on parameter contexts, and hence are modeled using the category of natural numbers (regarded as finite sets) and all functions between them.

At first glance, the interpretation of an operation $\mathsf{O} : (p \mid m_1, \ldots, m_k)$ in a semantic model $X \in \mathbf{Set}^{\mathbb{N}}$ should be a function $X(m_1) \times \ldots \times X(m_k) \to X(p)$, since O takes p parameters. However, the operation O can be used in any context Δ that has at least p parameters, so the interpretation of $\mathsf{O} : (p \mid m_1, \ldots, m_k)$ on a model $X \in \mathbf{Set}^{\mathbb{N}}$ should instead be a family of functions, for every $n \in \mathbb{N}$,

$$X(n + m_1) \times \ldots \times X(n + m_k) \to X(n + p),$$

interpreting O when it is used in a context with $n + p$ parameters. For example, if a theory includes the operation $\mathsf{close} : (1 \mid 0)$ then it includes all of the following terms for $n \in \mathbb{N}$:

$$x : n \mid a_1, \ldots, a_{n+1} \vdash \mathsf{close}(a_{n+1}; x(a_1, \ldots, a_n)).$$

If we think of parameters as scopes, and the set $X(n)$ as the set of computations that close n scopes, then we see that to model close must actually require a family of functions $X(n) \to X(n + 1)$ indexed by $n \in \mathbb{N}$, since close only closes the last of possibly many scopes and then delegates the closing of any remaining scopes to its continuation.

Definition 4.1. Let Σ be a parameterized signature (Definition 3.1). A Σ-*structure* $\mathcal{X}$ is an $X \in \mathbf{Set}^{\mathbb{N}}$ equipped with, for each $\mathrm{O} : (p \mid m_1, \ldots, m_k)$ and $n \in \mathbb{N}$, a function

$$\mathrm{O}_{\mathcal{X},n} : X(n+m_1) \times \ldots \times X(n+m_k) \to X(n+p).$$

Given a Σ-structure $\mathcal{X}$, the interpretation of operations can be extended to all terms (generated by rules (14) and (15)). The interpretation of a term $x_1 : m_1, \ldots, x_k : m_k \mid a_1, \ldots, a_p \vdash t$ is a sequence of functions

$$[\![t]\!]_{\mathcal{X},n} : X(n+m_1) \times \ldots \times X(n+m_k) \to X(n+p)$$

indexed by $n \in \mathbb{N}$ defined by structural recursion as follows. A (computation) variable

$$x_1 : m_1, \ldots, x_k : m_k \mid a_1, \ldots, a_{m_i} \vdash x_i(a_1, \ldots, a_{m_i})$$

is interpreted by the sequence of product projections

$$\pi_i : X(n+m_1) \times \ldots \times X(n+m_i) \times \ldots \times X(n+m_k) \to X(n+m_i).$$

A term $\Gamma \mid \Delta, a_1, \ldots, a_p \vdash \mathrm{O}(a_1, \ldots, a_p;\ b_1 \ldots b_{m_1}.t_1,\ \ldots,\ b_1 \ldots b_{m_k}.t_k)$ is interpreted by the sequence of functions

$$\mathrm{O}_{\mathcal{X},n+|\Delta|} \circ \langle [\![t_1]\!]_{\mathcal{X},n}, \ldots, [\![t_k]\!]_{\mathcal{X},n} \rangle.$$

Let us make a connection between the substitution rule (16) and our modeling of terms as sequences of functions. For every $n \in \mathbb{N}$, from a term

$$x_1 : m_1, \ldots, x_k : m_k \mid a_1, \ldots, a_p \vdash t$$

we can obtain a term

$$y_1 : n+m_1, \ldots, y_k : n+m_k \mid b_1, \ldots, b_n, a_1, \ldots, a_p \vdash t', \tag{40}$$

where $t' = t[\ldots, (b_1, \ldots, b_n, c_1, \ldots, c_{m_i} \vdash y_i(b_1, \ldots, b_n, c_1, \ldots, c_{m_i}))/x_i, \ldots]$, replaces all instances of $x_i(d_1, \ldots, d_{m_i})$ by $y_i(b_1, \ldots, b_n, d_1, \ldots, d_{m_i})$ in t. Informally, t' is the computation t running with n additional resources, which are passed to the computation variables $y_1, \ldots, y_k$ to consume. The interpretations of t and t' are connected by the formula

$$[\![t']\!]_{\mathcal{X},s} = [\![t]\!]_{\mathcal{X},s+n}.$$

Definition 4.2. Let $\mathcal{T}$ be a parameterized theory over the signature Σ. A Σ-structure $\mathcal{X}$ is a *model* of $\mathcal{T}$ if for every equation $\Gamma \mid \Delta \vdash s = t$ in $\mathcal{T}$, and every $n \in \mathbb{N}$, we have an equality of functions $[\![\Gamma \mid \Delta \vdash s]\!]_{\mathcal{X},n} = [\![\Gamma \mid \Delta \vdash t]\!]_{\mathcal{X},n}$.

Proposition 4.3. *The derivable equality $=_{\mathcal{T}}$ (Figure 3) for a parameterized algebraic theory $\mathcal{T}$ is sound: If $L =_{\mathcal{T}} R$ is derivable, then $[\![L]\!]_{\mathcal{X}} = [\![R]\!]_{\mathcal{X}}$ in any $\mathcal{T}$-model $\mathcal{X}$.*

Proof notes. The proof goes by induction on the structure of derivations of $L =_{\mathcal{T}} R$ in Figure 3. The only non-trivial case is the first rule in Figure 3, which follows from the semantic counterpart to substitution (16):

$$[\![t[t_1/x_1, \ldots, t_l/x_l]]\!]_{\mathcal{X}} = [\![t]\!]_{\mathcal{X},|\Delta'|} \cdot \langle [\![t_1]\!]_{\mathcal{X}}, \ldots, [\![t_l]\!]_{\mathcal{X}} \rangle$$

and this property can be shown by induction on t. □

Remark 4.4. A more abstract view on models is based on enriched categories, since parameterized algebraic theories can be understood in terms of enriched Lawvere theories [15, 34, 37]. This is potentially useful because, by interpreting algebraic theories in different categories, we can combine the algebra structure with other structure, such as topological or order structure for recursion [1, Section 6], or make connections with syntactic categories [39].

We now outline the general notion of model in this enriched setting. First, recall that the category $\mathbf{Set}^{\mathbb{N}}$ has a "Day convolution" monoidal structure [6]:

$$(X \otimes Y)(n) = \sum_{m_1+m_2=n} X(m_1) \times Y(m_2)$$

making $\mathbf{Set}^{\mathbb{N}}$ a monoidal category. Having chosen this monoidal structure for the category $\mathbf{Set}^{\mathbb{N}}$, we can consider categories enriched in $\mathbf{Set}^{\mathbb{N}}$. We can interpret a parameterized algebraic theory $\mathcal{T}$ in any $\mathbf{Set}^{\mathbb{N}}$-enriched category C with products, powers, and copowers. A $\mathcal{T}$-model in C comprises an object $X \in C$ together with, for each $\mathrm{O} : (p \mid m_1 \dots m_k)$, a morphism

$$\mathbf{y}(p) \cdot \big([\mathbf{y}(m_1), X] \times \cdots \times [\mathbf{y}(m_k), X]\big) \to X,$$

making a diagram commute for each equation in $\mathcal{T}$. Here, we write $\mathbf{y}(m) \coloneqq \mathbb{N}(m, -)$, and $A \cdot X$ and $[A, X]$ for the copower and power in C.

The elementary notion of model (Definition 4.2) is recovered when we consider $\mathbf{Set}^{\mathbb{N}}$ enriched in itself, because for the symmetric monoidal closed structure on $\mathbf{Set}^{\mathbb{N}}$: $[\mathbf{y}(m), X](n) = X(n+m)$, and the copower $\cdot$ is the Day convolution $\otimes$. This framework subsumes the one with the functors $(\triangleright)$, $(\triangleleft)$ from (9) since $\triangleright X = \mathbf{y}(1) \otimes X$ and $\triangleleft X = [\mathbf{y}(1), X]$.

4.2 Free Models of Parameterized Algebraic Theories and Induced Monads

Strong monads are a widely used encapsulation of computational effects, starting from Moggi's work [23]. First-order algebraic theories have a close correspondence with monads. To go from a theory to a monad, one notes that the construction assigning the free model $T(X)$ of the theory to each set X has the structure of a monad. The finitary monads on $\mathbf{Set}$ are exactly those that arise in this way. This is also true in the enriched setting (e.g., [15]). In this subsection we flesh out that construction in the case of parameterized algebraic theories.

There is an evident notion of homomorphism applicable to Σ-structures and $\mathcal{T}$-models, and thus we can sensibly discuss Σ-structures and $\mathcal{T}$-models that are *free* over some collection $X \in \mathbf{Set}^{\mathbb{N}}$ of generators, as follows.

Definition 4.5. Consider a $\mathcal{T}$-model $\mathcal{Y}$ with carrier $Y \in \mathbf{Set}^{\mathbb{N}}$ and a morphism $\eta_X : X \to Y$ in $\mathbf{Set}^{\mathbb{N}}$. Then $\mathcal{Y}$ is *free on* X, if for any other model $\mathcal{Z}$ and any morphism $f : X \to Z$ in $\mathbf{Set}^{\mathbb{N}}$, there exists a unique homomorphism of models $\hat{f} : \mathcal{Y} \to \mathcal{Z}$, that extends f, meaning $\hat{f} \circ \eta_X = f$ in $\mathbf{Set}^{\mathbb{N}}$.

In Proposition 4.6 below we will show that a particular construction $F_{\mathcal{T}}X$ gives the free $\mathcal{T}$-model on X. We discuss an abstract description of $F_{\mathcal{T}}X$ below, but for convenience we first describe it concretely. Informally, we can construct $F_{\mathcal{T}}X \in \mathbf{Set}^{\mathbb{N}}$ by taking $F_{\mathcal{T}}X(n)$ to be the set of $=_{\mathcal{T}}$-equivalence classes of terms with parameter context $a_1, \dots, a_n$ whose m_i-ary computation variables come from $X(m_i)$. In more detail, elements of $F_{\mathcal{T}}X(n)$ are represented by triples $\langle \Gamma; [t]; \vec{c} \rangle$ consisting of

— a computation context $\Gamma = x_1 : m_1, \dots, x_k : m_k$,
— an $=_{\mathcal{T}}$-equivalence class of terms $[\Gamma \mid a_1, \dots, a_n \vdash t]_{=_{\mathcal{T}}}$ in computation context Γ and parameter context $a_1, \dots, a_n$, and
— a list $\vec{c} = c_1, \dots, c_k$ where $c_i \in X(m_i)$.

Then $F_{\mathcal{T}}X(n)$ consists of the $\sim$-equivalence classes of these triples,

$$F_{\mathcal{T}}X(n) = \{\langle \Gamma = x_1 : m_1, \dots, x_k : m_k; [\Gamma \mid a_1, \dots, a_n \vdash t]_{=_{\mathcal{T}}}; c_1, \dots, c_k \rangle \mid c_i \in X(m_i)\}/\sim, \tag{41}$$

where $\sim$ is the equivalence relation generated by the following basic cases:

— α-renaming: $\langle \Gamma; [t]; \vec{c} \rangle \sim \langle \Gamma'; [t']; c \rangle$ where $\Gamma' \mid \vec{a} \vdash t'$ is obtained from $\Gamma \mid \vec{a} \vdash t$ by renaming the variables in Γ.

—Permutation: $\langle \Gamma; [t]; \vec{c} \rangle \sim \langle \Gamma'; [t']; \vec{c'} \rangle$, where $\Gamma' \mid \vec{a} \vdash t'$ is the result of a permutation of the list Γ, and $\vec{c'}$ is the result of the same permutation applied to the list $\vec{c}$.
—Contraction: $\langle \Gamma, x : m, y : m; [t]; \vec{c}, d, d \rangle \sim \langle \Gamma, x : m; [t[x/y]]; \vec{c}, d \rangle$.
—Weakening: $\langle \Gamma; [t]; \vec{c} \rangle \sim \langle \Gamma, x : m; [t]; \vec{c}, d \rangle$, where $d \in X(m)$ and the $[t]$ on the right is understood to be in the weakened context.

It is straightforward to make $F_{\mathcal{T}}X$ into a Σ-structure.

As usual, we can also describe $F_{\mathcal{T}}X$ abstractly. Let $\mathcal{A}$ be the full subcategory of $\mathbf{Set}^{\mathbb{N}}$ whose objects are finite sums of representables. For each context $\Gamma = x_1 : m_1, \ldots, x_k : m_k$, consider the object $V_\Gamma \in \mathbf{Set}^{\mathbb{N}}$ where

$$V_\Gamma(n) = \{\Gamma \mid b_1, \ldots, b_n \vdash x_i(b_1, \ldots, b_n)\}. \tag{42}$$

(So $|V_\Gamma(n)| = |\{1 \le i \le k \mid m_i = n\}|$.) Then $V_\Gamma \in \mathcal{A}$, because $V_\Gamma \cong \mathbf{y}(m_1) + \ldots + \mathbf{y}(m_k)$, where as above $\mathbf{y}(m)$ is the representable $\mathbb{N}(m, -)$. Indeed, every object of $\mathcal{A}$ is isomorphic to some V_Γ. The morphisms $\alpha : V_\Gamma \to V_{\Gamma'}$ of $\mathcal{A}$ correspond to context substitutions from Γ to Γ' generated by renamings, permutations, contractions, and weakenings.

Let $T_{\mathcal{T}} : \mathcal{A} \to \mathbf{Set}^{\mathbb{N}}$ be a functor where

$$T_{\mathcal{T}}(V_\Gamma)(n) = \{[\Gamma \mid a_1, \ldots, a_n \vdash t]_{=_{\mathcal{T}}}\}$$

is the set of $=_{\mathcal{T}}$-equivalence classes of terms $\Gamma \mid a_1, \ldots, a_n \vdash t$, and the action of $T_{\mathcal{T}}$ on morphisms is given by applying context substitutions. (This determines T on all of $\mathcal{A}$ up to natural isomorphism.) Then $F_{\mathcal{T}}$ is the left Kan extension of $T_{\mathcal{T}}$ along the inclusion functor $\mathsf{inc} : \mathcal{A} \to \mathbf{Set}^{\mathbb{N}}$. As usual, the left Kan extension has a coend formula:

$$F_{\mathcal{T}}X(n) \cong \int^{V \in \mathcal{A}} T_{\mathcal{T}}V(n) \times \mathbf{Set}^{\mathbb{N}}(V, X).$$

In the case where $V = V_\Gamma$, we have $\mathbf{Set}^{\mathbb{N}}(V_\Gamma, X) \cong X(m_1) \times \ldots \times X(m_k)$, from which the reader familiar with coends will be able to recover the concrete presentation (41). As usual, a left Kan extension along a full and faithful functor restricts back to the original functor, so that

$$F_{\mathcal{T}}(V_\Gamma)(n) \cong T_{\mathcal{T}}(V_\Gamma)(n) = \{[\Gamma \mid a_1, \ldots, a_n \vdash t]_{=_{\mathcal{T}}}\} \tag{43}$$

for any context Γ, as may also be verified from the concrete description.

Proposition 4.6.

(1) $F_{\mathcal{T}}X$ is a $\mathcal{T}$-model, and moreover a free $\mathcal{T}$-model over X.
(2) $F_{\mathcal{T}}$ extends to a monad on $\mathbf{Set}^{\mathbb{N}}$, strong for the Day tensor.
(3) The derivable equality $(=_{\mathcal{T}})$ in a parameterized algebraic theory $\mathcal{T}$ is complete: If an equation is valid in every $\mathcal{T}$-model, then it is derivable in $=_{\mathcal{T}}$.

A monad T on $\mathbf{Set}^{\mathbb{N}}$ strong for the Day tensor is a monad in the usual sense equipped with a strength $X \otimes TY \to T(X \otimes Y)$, where $\otimes$ is the Day tensor defined in Remark 4.4.

Proof. The first part of (1) is straightforward and standard: The interpretation of operation symbols is read off from (15). For the second part, we use the variable introduction rule (14) to define $\eta_X : X \to F_{\mathcal{T}}X$ by sending $c \in X(n)$ to the class of

$$\langle x : n; [x : n \mid a_1, \ldots, a_n \vdash x(a_1, \ldots, a_n)]_{=_{\mathcal{T}}}; c \rangle.$$

Then, for any $\mathcal{T}$-model $\mathcal{Z}$ with carrier Z and map $\phi : X \to Z$, we extend ϕ (uniquely) to a homomorphism $F_{\mathcal{T}}X \to \mathcal{Z}$ by sending an element of $F_{\mathcal{T}}X(n)$ represented by

$$\langle x_1 : m_1, \ldots, x_k : m_k; [x_1 : m_1, \ldots, x_k : m_k \mid a_1, \ldots a_n \vdash t]_{=_{\mathcal{T}}}; c_1, \ldots, c_k \rangle$$

to

$$[\![\vec{x} \mid \vec{a} \vdash t]\!]_{\mathcal{Z},0}(\phi_{m_1}(c_1), \ldots, \phi_{m_k}(c_k)) \in Z(n).$$

For Claim (3), we use the models $F_{\mathcal{T}}(V_\Gamma)$, using the description in (43). For any term $\Gamma \mid a_1 \ldots, a_p \vdash t$, its interpretation in the model $F_{\mathcal{T}}(V_\Gamma)$ gives function

$$[\![t]\!]_0 : F_{\mathcal{T}}(V_\Gamma)(m_1) \times \ldots \times F_{\mathcal{T}}(V_\Gamma)(m_k) \to F_{\mathcal{T}}(V_\Gamma)(p).$$

One can check that applying $[\![t]\!]_0$ to the tuple

$$\langle [x_1(a_1, \ldots, a_{m_1})]_{=_{\mathcal{T}}}, \ldots, [x_k(a_1, \ldots, a_{m_k})]_{=_{\mathcal{T}}} \rangle$$

of inclusions of variables gives the $=_{\mathcal{T}}$-equivalence of t as an element of $F_{\mathcal{T}}(V_\Gamma)(p)$. Thus if u is another term in the same context and $[\![t]\!] = [\![u]\!]$ in the model $F_{\mathcal{T}}(V_\Gamma)$, then $t =_{\mathcal{T}} u$ is derivable.

For Claim (2), the η_X defined above becomes the monadic unit, and bind is defined in terms of substitution as usual (rule (16)). By the general properties of the Day tensor product, the strength is determined by a natural transformation

$$X(p) \otimes F_{\mathcal{T}}Y(n) \to F_{\mathcal{T}}(X \otimes Y)(p + n).$$

This map sends $c \in X(p)$ paired with an element represented by

$$\langle y_1 : m_1, \ldots, y_k : m_k; [y_1 : m_1, \ldots, y_k : m_k \mid b_1, \ldots b_n \vdash t]_{=_{\mathcal{T}}}; d_1, \ldots, d_k \rangle,$$

where $d_i \in Y(m_i)$ to the class of

$$\langle y_1 : p + m_1, \ldots, y_k : p + m_k; \\ [y_1 : p + m_1, \ldots, y_k : p + m_k \mid a_1, \ldots, a_p, b_1, \ldots b_n \vdash t']_{=_{\mathcal{T}}}; \\ \langle \langle p, m_1 \rangle, \langle c, d_1 \rangle \rangle, \ldots, \langle \langle p, m_k \rangle, \langle c, d_k \rangle \rangle \rangle,$$

where $t' = t[\ldots, (a_1, \ldots, a_p, c_1, \ldots, c_{m_i} \vdash y_i(a_1, \ldots, a_p, c_1, \ldots, c_{m_i}))/y_i, \ldots]$ and the $\langle \langle p, m_i \rangle, \langle c, d_i \rangle \rangle$ are elements of $(X \otimes Y)(p+m_i) = \sum_{h_1+h_2=p+m_i} X(h_1) \times Y(h_2)$ that lie within the $X(p) \times Y(m_i)$ summand. It is straightforward to check the required laws. □

In Section 4.4 we will consider explicit syntax-free characterizations of the free models for particular scoped theories.

Remark 4.7. We mention that $F_{\mathcal{T}}$ is part of an equivalence between monads on $\mathbf{Set}^{\mathbb{N}}$ strong for the Day tensor and whose functor part preserves sifted colimits, and parameterized algebraic theories. This result can be proved using similar methods to those employed for parameterized theories with unrestricted parameters [37, Corollary 1] and linear parameters [40, Section 5]. We omit the proof as this result will not be used in the rest of this article.

4.3 Set-Monads Induced by Scoped Theories in General

The previous subsection introduced a way of building a monad $F_{\mathcal{T}}$ from a parameterized algebraic theory $\mathcal{T}$, via its free models. We now relate this with earlier work on monads for scoped effects.

Recall from Section 2.2 (13) that the functors $\uparrow : \mathbf{Set} \to \mathbf{Set}^{\mathbb{N}}$ and $\downarrow : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set}$ form an adjunction. The monad $F_{\mathcal{T}}$ induces a monad $F'_{\mathcal{T}}$ on $\mathbf{Set}$:

$$F'_{\mathcal{T}} \coloneqq \downarrow \circ F_{\mathcal{T}} \circ \uparrow. \tag{44}$$

In this section we analyze this monad $F'_{\mathcal{T}}$ on $\mathbf{Set}$ in the case where the parameterized algebraic signature of $\mathcal{T}$ is induced by a scoped signature, i.e., some (first-order) algebraic operations plus some scoped operations, according to Example 3.3.

—In Proposition 4.8 we show that the induced monad $F'_{\mathcal{T}}$ on $\mathbf{Set}$ accommodates the algebraic and scoped operations from the scoped signature.

—In Proposition 4.9 we characterize the induced monad $F'_{\mathcal{T}}$ on **Set** in the case where there are no scoped operations, but where there may be equations. It is the monad induced by models of the underlying first-order algebraic theory.
—In Theorem 4.10 we characterize the induced monad $F'_{\mathcal{T}}$ on **Set** in the case where there may be scoped operations but there are no equations. We show that this induced monad $F'_{\mathcal{T}}$ is isomorphic to the monad of Proposition 2.14 that was introduced by Piróg et al. [25].

Later, in Sections 4.4 and 4.5 we will go beyond this, to consider settings where there are both scoped operations *and* equations. The constructions of this section are general, and we return to examples in Section 4.4.

4.3.1 *Algebraic and Scoped Operations on the Induced Set-Monad*

Proposition 4.8. *Let $\mathcal{T} = \langle \Sigma, E \rangle$ be a parameterized algebraic theory where the signature Σ arises from a pair of signatures for algebraic and scoped operations by the construction in Example 3.3.*

(1) For each algebraic operation $(o : k)$, the monad $F'_{\mathcal{T}}$ supports an algebraic operation of the same arity, in the sense of Definition 2.11.
(2) For every scoped operation $(s : k)$, the monad $F'_{\mathcal{T}}$ supports a scoped operation of the same arity, in the sense of Definition 2.12, i.e., a natural transformation $(F'_{\mathcal{T}}(-))^k \to F'_{\mathcal{T}}(-)$.

Proof. We have a natural transformation $(F_{\mathcal{T}}X)^k \to F_{\mathcal{T}}X$ given by substituting terms into the operation o and this is "algebraic" in the sense of commuting with postcomposition by the monadic multiplication, so (1) follows easily. In general, an operation $s : (0 \mid 1, \dots, 1)$ appears as a natural transformation $(\triangleleft F_{\mathcal{T}}X)^k \to F_{\mathcal{T}}X$. However, it is easy to show that $\triangleleft$ preserves $\mathcal{T}$-models, and in the setting of (2) we can use $\mathsf{close} : (1 \mid 0)$ to get a map

$$X \xrightarrow{\eta_X} F_{\mathcal{T}}X \xrightarrow{[\![\mathsf{close}]\!]} \triangleleft F_{\mathcal{T}}X.$$

By the universal property of free models, this induces another map $F_{\mathcal{T}}X \to \triangleleft F_{\mathcal{T}}X$. This map adds a close at the leaves of each syntax tree, as opposed to at the root. Precomposing the scoped operation $(\triangleleft F_{\mathcal{T}}X)^k \to F_{\mathcal{T}}X$ with k copies of $F_{\mathcal{T}}X \to \triangleleft F_{\mathcal{T}}X$, we have a natural transformation $(F_{\mathcal{T}}X)^k \to F_{\mathcal{T}}X$ which restricts to the desired scoped operation on $F'_{\mathcal{T}}$. □

4.3.2 *Characterization of the Induced Set-Monad in the Absence of Scoped Operations.*

Consider an ordinary algebraic theory $\mathcal{T} = \langle \Sigma, E \rangle$ with a signature and equations. This ordinary theory induces a parameterized algebraic theory $\mathcal{T}_2$, following Example 3.3, when it is regarded as a scoped theory without any scoped operations. In Proposition 4.9, we show that these two theories induce the same monad on **Set**.

In full detail, the parameterized theory $\mathcal{T}_2$ is given as follows. We consider a scoped signature with the algebraic operations of $\mathcal{T}$ and no scoped operations. This scoped signature induces a parameterized algebraic signature, according to Example 3.3. This parameterized signature has an operation of arity $(0 \mid 0, \dots, 0)$ for every operation of the ordinary theory, together with an operation $\mathsf{close} : (1 \mid 0)$. The equations of the ordinary theory $t = u$ can be regarded as equations of the parameterized algebraic signature, $t^0 = u^0$, according to the translation $(-)^0$ from Section 3.1. Thus the ordinary algebraic theory $\mathcal{T}$ induces a parameterized algebraic theory $\mathcal{T}_2$.

As a mild digression, we remark that since there are no scoped operations, it is natural to omit the close operation. This gives rise to a simpler parameterized algebraic theory, which we call $\mathcal{T}_1$, and which is defined in the same way as $\mathcal{T}_2$ but without the close operation. We show that the induced monads on **Set** are all the same.

Proposition 4.9. *Let $\mathcal{T} = \langle \Sigma, E \rangle$ be an algebraic theory that induces a monad T on* **Set**. *Then T is isomorphic to the* **Set** *restriction of the monads induced by the parameterized theory $\mathcal{T}_1$ and by the parameterized theory $\mathcal{T}_2$:*

$$T \cong F'_{\mathcal{T}_1} \cong F'_{\mathcal{T}_2}$$

with $F'_{\mathcal{T}_1}$ and $F'_{\mathcal{T}_2}$ monads on **Set** *as in (44).*

Proof. We will prove a stronger statement about each theory. The basic observation to make about $\mathcal{T}_1$ is that a $\mathcal{T}_1$-algebra structure on $Y \in \mathbf{Set}^{\mathbb{N}}$ is equivalent to a $\mathcal{T}$-algebra structure on each $Y(n)$ (with no conditions relating the $\mathcal{T}$-algebras for different n). Hence, it is easy to see that for any $X \in \mathbf{Set}^{\mathbb{N}}$, $(F_{\mathcal{T}_1} X)(n) \cong T(X(n))$. Then by definition of the functors $\restriction$ and $\downarrow$ we obtain $F'_{\mathcal{T}_1}(A) \cong T(A)$ for any set A.

For $\mathcal{T}_2$, the basic observation is that a $\mathcal{T}_2$-algebra structure on $Y \in \mathbf{Set}^{\mathbb{N}}$ is just a $\mathcal{T}_1$-algebra structure (levelwise $\mathcal{T}$-algebra) together with a sequence of functions $\mathsf{close}_{Y,n} : Y(n) \to Y(n+1)$. Hence it is routine to check that the free algebra on $X \in \mathbf{Set}^{\mathbb{N}}$ can be given by the recurrence $F_{\mathcal{T}_2}(X)(0) = T(X(0))$ and $F_{\mathcal{T}_2}(X)(n+1) = T(X(n) + F_{\mathcal{T}_2}(X)(n))$. In particular, for a set A, the free model $F_{\mathcal{T}_2}(\restriction A)$ is given by $\bar{T}A \coloneqq \lambda n.\, T^{n+1}A$, with the obvious interpretation of the algebraic operations in Σ, and $\mathsf{close}_{\bar{T}A,n} : T^{n+1}A \to T^{n+2}A$ given by the monadic unit $\eta_{T^{n+1}A}$. □

Proposition 4.9 establishes the connection between monads induced by parameterized theories with monads for ordinary algebraic theories. As we can already see with our two theories $\mathcal{T}_1$ and $\mathcal{T}_2$, there can be multiple parameterized theories inducing the same ordinary algebraic theory. We include the theory $\mathcal{T}_1$ in the proposition above because it is the more natural way to view $\mathcal{T}$ as a parameterized theory. We can demonstrate its significance further using the abstract situation of Remark 4.4: We can make **Set** into an appropriate $\mathbf{Set}^{\mathbb{N}}$ enriched category by change-of-enrichment along the diagonal functor $\mathbf{Set} \to \mathbf{Set}^{\mathbb{N}}$, following which it is easy to see that $\mathcal{T}_1$-models in **Set** are equivalent to $\mathcal{T}$-models in **Set**. Models of $\mathcal{T}_2$ in **Set** are also $\mathcal{T}$-models, but equipped with an endomorphism of the carrier coming from the interpretation of close. However, $\mathcal{T}_2$ is our starting point for adding scoped operations to an algebraic theory, as we saw in the examples of Section 3.2 and will return to in Section 4.5.

4.3.3 Characterization of the Induced Set-Monad in the Absence of Equations. We turn now to the setting of a scoped signature, with no equations. We show that the monad $F'_{\mathcal{T}}$ on **Set** is the same as the monad of Proposition 2.14 given by Piróg et al. [25]. We do this by characterizing $F_{\mathcal{T}}$ when $\mathcal{T}$ has no equations, as follows.

Theorem 4.10. *Consider signatures for algebraic Σ and scoped Σ' effects with no equations, inducing a parameterized algebraic theory $\mathcal{T}$ (via Example 3.3). We have an isomorphism of monads*

$$F_{\mathcal{T}} \cong \left(\bar{\Sigma} + \left(\bar{\Sigma}' \circ \triangleleft\right) + \triangleright\right)^*.$$

Here $\triangleleft, \triangleright$ are endofunctors on $\mathbf{Set}^{\mathbb{N}}$ as in (9), and $\bar{\Sigma}$ and $\bar{\Sigma}'$ are the endofunctors on $\mathbf{Set}^{\mathbb{N}}$ corresponding to the signatures Σ, Σ', as given in (10).

Proof. Denote by G the functor $\left(\bar{\Sigma} + \left(\bar{\Sigma}' \circ \triangleleft\right) + \triangleright\right)$. We can think of the elements of $G^*(X)(n)$ as trees with nodes being operations from Σ, Σ', or close and leaves from X at an appropriate index. Consider the description of $F_{\mathcal{T}} X(n)$ from Equation (41) in terms of equivalence classes. A representative of an equivalence class has the form:

$$\langle \Gamma = x_1 : m_1, \ldots, x_k : m_k;\ \Gamma \mid a_1, \ldots, a_n \vdash t;\ c_1, \ldots, c_k \rangle,$$

where each $c_i \in X(m_i)$. We define a natural transformation

$$\psi : F_{\mathcal{T}} X \to G^*(X)$$

by induction on terms $\Gamma \mid a_1, \ldots, a_n \vdash t$. The tuple $\langle \Gamma;\ x_i(a_1, \ldots, a_{m_i});\ \vec{c} \rangle$ is mapped to the leaf c_i. If t starts with a scoped operation from Σ', meaning $t = \mathsf{sc}(a_{n+1}.t_1, \ldots, a_{n+1}.t_{ar(\mathsf{sc})})$, we use the trees

$$\psi_{n+1}\langle \Gamma;\ \Gamma \mid a_1, \ldots, a_n, a_{n+1} \vdash t_j;\ \vec{c} \rangle \in \triangleleft G^*(X)(n) = G^*(X)(n+1)$$

as children to construct a tree with root sc. Recall that the functor $\triangleleft$ increases the indexing by 1, and $\triangleright$ decreases it. If t is an algebraic operation from Σ, then ψ is defined similarly (without using $\triangleleft$). If $t = \mathsf{close}(a_n; t')$, then $\psi_{n-1}\langle \Gamma;\ \Gamma \mid a_1, \ldots, a_{n-1} \vdash t';\ \vec{c} \rangle \in G^*(X)(n-1)$ becomes the child of the root labeled close. It is easy to check that ψ respects the equivalence relation generated by α-renaming, permutation, weakening, and contraction of Γ, using the fact that these operations have the corresponding effect on $\vec{c}$ and therefore on the trees in the image of ψ.

To check that ψ is an isomorphism, we can use the obvious candidate inverse, mapping trees from $G^*(X)$ to terms with free computation variables from X. The action of G^* on $\phi : X \to Y$ is to apply ϕ to the leaves of the tree, while $F_{\mathcal{T}}(\phi)$ replaces $\vec{c}$ in a tuple with $\phi_{m_1}(c_1), \ldots, \phi_{m_k}(c_k)$. Therefore ψ is natural in X.

To show ψ is an isomorphism of monads, we also need to show $\psi \circ \eta = \eta$, which is true because $\eta : X \to F_{\mathcal{T}}(X)$ maps $c \in X(m)$ to the tuple $\langle \Gamma = x : m;\ \Gamma \mid a_1, \ldots, a_m \vdash x(a_1, \ldots, a_m);\ c \rangle$, and $\eta : X \to G^*(X)$ maps c to the leaf c. We also need to show $\psi \circ \mu = \mu \circ \psi G^* \circ F_{\mathcal{T}}\psi$, which follows if we observe that multiplication for $F_{\mathcal{T}}$ is given by simultaneous term substitution, while for G^* multiplication replaces the leaves of a tree with trees. □

4.4 Reconstructing Models for Scoped Effects via Parameterized Theories: Examples

In this section we revisit the examples of parameterized algebraic theories from Section 3.2: nondeterminism with once, catching exceptions, state, and nondeterminism with cut. We show that the free models in the sense of Section 4.2 correspond exactly with the earlier scoped models of Piróg et al. [25].

To characterize them as certain free models of parameterized algebraic theories, we need the following notion.

Definition 4.11. $X \in \mathbf{Set}^{\mathbb{N}}$ is *truncated* if $X(n+1) = \emptyset$ for all $n \in \mathbb{N}$.

Equivalently, X is truncated if $X = \restriction(X(0))$. The free model on a truncated X corresponds to the case where computation variables can only denote programs with no open scopes. This is the case in the development of Piróg et al. [25], where if the programmer opens a scope, a matching closing of the scope is implicitly part of the program.

4.4.1 Nondeterminism with once. Recall the parameterized theory for nondeterminism with once (signature in Example 3.2 and equations in Figure 4). It follows from Proposition 4.6 that this theory has a free model on each X in $\mathbf{Set}^{\mathbb{N}}$, with carrier denoted by $T_{\mathbf{o}}(X) \in \mathbf{Set}^{\mathbb{N}}$. For X truncated, the free model on X has an elegant description:

$$T_{\mathbf{o}}(X)(n) = \mathsf{List}^{n+1}(X(0)).$$

Here $\mathsf{List}(A)$ is the set of lists over A, so the n-layer nested lists $\mathsf{List}^{n+1}(X(0))$ can be thought of as the set of balanced trees of depth exactly $n+1$ but with arbitrary degree, and leaves labeled by $X(0)$, where we distinguish between nodes with zero degree and leaf nodes.

In this case the interpretation of once chooses the first element of a list and closing a scope wraps its continuation as a singleton list. Choice is interpreted as list concatenation ($+\!\!+$), and failure as

the empty list []:

$$\begin{aligned}
&\mathsf{once}_n : T_{\mathbf{o}}(X)(n+1) \to T_{\mathbf{o}}(X)(n) && \mathsf{once}_n([]) = [],\ \mathsf{once}_n([x, \ldots]) = x \\
&\mathsf{close}_n : T_{\mathbf{o}}(X)(n) \to T_{\mathbf{o}}(X)(n+1) && \mathsf{close}_n(x) = [x] \\
&\mathsf{or}_n : T_{\mathbf{o}}(X)(n) \times T_{\mathbf{o}}(X)(n) \to T_{\mathbf{o}}(X)(n) && \mathsf{or}_n(x_1, x_2) = x_1 +\!\!+ x_2 \\
&\mathsf{fail}_n : 1 \to T_{\mathbf{o}}(X)(n) && \mathsf{fail}_n() = [].
\end{aligned}$$

In fact the model $T_{\mathbf{o}}(X)$ we just described is the same as the model for nondeterminism of Piróg et al. [25, Example 4.2].

Theorem 4.12. *For a truncated $X \in \mathbf{Set}^{\mathbb{N}}$, $T_{\mathbf{o}}(X)$ is the free model of the parameterized theory of nondeterminism with* once *(Example 3.5 and Figure 4). Moreover, the model for nondeterminism with* once *from [25, Example 4.2], starting from a set A, is the free model on $\restriction A \in \mathbf{Set}^{\mathbb{N}}$.*

Proof notes. It is easy to show that the operations on $T_{\mathbf{o}}(X)$ defined in this section satisfy the equations in Figure 4, and therefore $T_{\mathbf{o}}(X)$ is a model. Denote by $F_{\mathbf{o}}(X)$ the free model in the sense of Proposition 4.6. We apply the definition of freeness to the map $X \to T_{\mathbf{o}}(X)$ that sends each $c \in X(0)$ to the singleton list $[c]$, to obtain a homomorphism of models $\rho : F_{\mathbf{o}}(X) \to T_{\mathbf{o}}(X)$. To show $T_{\mathbf{o}}(X)$ is free, we show that ρ has an inverse $\sigma : T_{\mathbf{o}}(X) \to F_{\mathbf{o}}(X)$.

Recall from Equation (41) that $F_{\mathbf{o}}(X)(n)$ contains equivalence classes of terms in context, with computation variables taken from X. For simplicity, assume that $X(0)$ is finite. Then we define σ using representatives of the equivalence classes where the context Γ_X contains a variable $(x : 0)$ for each element of $X(0)$. We define σ by induction:

$$\begin{aligned}
&\sigma_n([]) = \Gamma_X \mid a_1, \ldots, a_n \vdash \mathsf{fail} \\
&\sigma_0(\mathsf{x} :: \mathsf{xs}) = \Gamma_X \mid - \vdash \mathsf{or}(\mathsf{x}, \sigma_0(\mathsf{xs})) \\
&\sigma_{n+1}(\mathsf{x} :: \mathsf{xs}) = \Gamma_X \mid a_1, \ldots, a_{n+1} \vdash \mathsf{or}\big(\mathsf{close}(a_{n+1}; \sigma_n(\mathsf{x})), \sigma_{n+1}(\mathsf{xs})\big).
\end{aligned}$$

If $X(0)$ is infinite, $\sigma_n(\mathsf{xs})$ is defined similarly, but Γ_X only contains variables corresponding to the elements of $X(0)$ that actually appear in xs. The terms in the image of σ are essentially normal forms for the parameterized theory, when computation variables only have arity 0.

We show that σ is a homomorphism of models, meaning it commutes with the operations in the theory, using the equations in Figure 4. The case for once also requires an induction on lists.

To show $\rho \circ \sigma = \mathsf{id}_{T_{\mathbf{o}}(X)}$, we use induction on $n \in \mathbb{N}$, followed by induction on lists, and the fact that ρ is a homomorphism. To show $\sigma \circ \rho = \mathsf{id}_{F_{\mathbf{o}}(X)}$, we use induction on the term formation rules, and use the fact that σ is a homomorphism. □

4.4.2 Exceptions. Recall the parameterized theory of throwing and catching exceptions in Example 3.6 and (24)–(26). For truncated $X \in \mathbf{Set}^{\mathbb{N}}$, the free model of the theory of exceptions has carrier:

$$T_{\mathbf{c}}(X)(n) = X(0) + \{e_0, \ldots, e_n\},$$

where e_{n-i} corresponds to the term (in normal form) that closes i scopes then throws.

To define the operations catch_n and close_n we pattern match on the elements of $T_{\mathbf{c}}(X)(n+1)$ using the isomorphism $T_{\mathbf{c}}(X)(n+1) \cong T_{\mathbf{c}}(X)(n) + \{e_{n+1}\}$ and elide the coproduct injections. Below, x is an element of $T_{\mathbf{c}}(X)(n)$, standing for a computation in normal form:

$$\begin{aligned}
&\mathsf{catch}_n : T_{\mathbf{c}}(X)(n+1) \times T_{\mathbf{c}}(X)(n+1) \to T_{\mathbf{c}}(X)(n) \\
&\mathsf{catch}_n(x, _) = x,\ \mathsf{catch}_n(e_{n+1}, x) = x,\ \mathsf{catch}_n(e_{n+1}, e_{n+1}) = e_n
\end{aligned}$$

$$\begin{aligned}
&\mathsf{close}_n : T_{\mathbf{c}}(X)(n) \to T_{\mathbf{c}}(X)(n+1) && \mathsf{close}_n(x) = x \\
&\mathsf{throw}_n : 1 \to T_{\mathbf{c}}(X)(n) && \mathsf{throw}_n() = e_n.
\end{aligned}$$

The cases in the definition of catch_n correspond to Equations (24)–(26), respectively. In the third case, an exception inside $n+1$ scopes in the second argument of catch becomes an exception inside n scopes.

Theorem 4.13. *For a truncated $X \in \mathbf{Set}^{\mathbb{N}}$, $T_{\mathbf{c}}(X)$ is the free model for the parameterized theory of exceptions (24)–(26). The model for exception catching from [25, Example 4.5], which starts from a set A, is the free model on $\uparrow A \in \mathbf{Set}^{\mathbb{N}}$.*

Proof notes. We follow the same outline as for the proof of Theorem 4.12. Consider the map $X \to T_{\mathbf{c}}(X)$ that maps each element of $X(0)$ to itself. This has a unique extension to a map out of the free model on X, $\rho : F_{\mathbf{c}}(X) \to T_{\mathbf{c}}(X)$. We define a candidate inverse $\sigma : T_{\mathbf{c}}(X) \to F_{\mathbf{c}}(X)$ to ρ. It suffices to check being inverse as morphisms in $\mathbf{Set}^{\mathbb{N}}$, since then σ is automatically a homomorphism. Using the intuition of normal forms and the isomorphism $T_{\mathbf{c}}(X)(n) \cong X(0) + \{e_0, \dots, e_n\}$:

$$\sigma_n(x) = x : 0 \mid a_1, \dots, a_n \vdash \mathsf{close}(a_n; \dots \mathsf{close}(a_1; x) \dots)$$
$$\sigma_n(e_{n-i}) = - \mid a_1, \dots, a_n \vdash \mathsf{close}(a_{n+1}; \dots \mathsf{close}(a_{n-i+1}; \mathsf{throw}) \dots), \quad \text{for each } 0 \le i \le n.$$

If we consider an element $z \in T_{\mathbf{c}}(X)(n)$, and the isomorphism $T_{\mathbf{c}}(X)(n+1) \cong T_{\mathbf{c}}(X)(n) + \{e_{n+1}\}$, the following equation holds:

$$\sigma_{n+1}(z) = \Gamma_{\sigma_n(z)} \mid a_1, \dots, a_{n+1} \vdash \mathsf{close}(a_{n+1}; \sigma_n(z)). \tag{45}$$

To show $\rho \circ \sigma = \mathsf{id}_{T_{\mathbf{c}}(X)}$ we use that ρ is a homomorphism. For $\sigma \circ \rho = \mathsf{id}_{F_{\mathbf{c}}(X)}$, we proceed by induction on the term formation rules. We use Equation (45), and in the catch case we also use the Equations (24) to (26) from the parameterized theory. □

4.4.3 State with Local Values. Recall the parameterized theory of mutable state with local values in Example 3.8 and its Equations (27)–(33). The free model, in the sense of Proposition 4.6, on a truncated $X \in \mathbf{Set}^{\mathbb{N}}$ has carrier:

$$T_{\mathbb{l}}(X)(0) = \big(\mathbb{2} \Rightarrow X(0) \times \mathbb{2}\big) \qquad T_{\mathbb{l}}(X)(n+1) = \big(\mathbb{2} \Rightarrow T_{\mathbb{l}}(X)(n)\big).$$

The operations on this model are

$$\begin{aligned}
&\mathsf{local}^i_n : T_{\mathbb{l}}(X)(n+1) \to T_{\mathbb{l}}(X)(n) && \mathsf{local}^i_n(f) = (f\ i)\\
&\mathsf{close}_n : T_{\mathbb{l}}(X)(n) \to T_{\mathbb{l}}(X)(n+1) && \mathsf{close}_n(f) = \lambda s.\, f\\
&\mathsf{put}^i_n : T_{\mathbb{l}}(X)(n) \to T_{\mathbb{l}}(X)(n) && \mathsf{put}^i_n(f) = \lambda s.\, f\ i\\
&\mathsf{get}_n : T_{\mathbb{l}}(X)(n)^2 \to T_{\mathbb{l}}(X)(n) && \mathsf{get}_n(f,g) = \lambda s. \begin{cases} f\ s & s = 0 \\ g\ s & \text{otherwise} \end{cases}.
\end{aligned}$$

Notice that the continuation of local^i uses the new state i, whereas close discards the state s which comes from the scope that is being closed.

If we only consider Equations (27)–(32), omitting (33), the carrier of the free model on a truncated $X \in \mathbf{Set}^{\mathbb{N}}$ is.

$$T'_{\mathbb{l}}(X)(0) = \big(\mathbb{2} \Rightarrow X(0) \times \mathbb{2}\big) = T_{\mathbb{l}}(X)(0) \qquad T'_{\mathbb{l}}(X)(n+1) = \big(\mathbb{2} \Rightarrow T'_{\mathbb{l}}(X)(n) \times \mathbb{2}\big). \tag{46}$$

In fact, $T'_{\mathbb{l}}(X)$ is the model of state with local proposed in [25, Section 7.1].

Theorem 4.14. *Given a truncated $X \in \mathbf{Set}^{\mathbb{N}}$, $T_{\mathbb{l}}(X)$ is the free model of the parameterized theory of state with local values, Equations (27) to (33). Moreover, $T'_{\mathbb{l}}(X)$ is the free model for Equations (27) to (32).*

Consider the example of state with local variables from Piróg et al. [25, Section 7.1], specialized to one memory location storing one bit, reading the return type (written "a" in [25]) as a set A.

The model proposed in [25, Section 7.1] is the free model on $\restriction A$ for the parameterized theory with Equations (27)–(32).

Proof notes. We show that $T_{\mathsf{l}}(X)$ is the free model following the same proof outline as for Theorems 4.12 and 4.13. It is easy to equip $T'_{\mathsf{l}}(X)$ with operations and do a similar proof of freeness.

Consider the map $X \to T_{\mathsf{l}}(X)$ that sends each $x \in X(0)$ to the function $\lambda s.\,(x, s)$, and let $\rho : F_{\mathsf{l}}(X) \to T_{\mathsf{l}}(X)$ be its unique extension out of the free model. We define a candidate inverse for ρ using the intuition of normal forms, $\sigma : T_{\mathsf{l}}(X) \to F_{\mathsf{l}}(X)$:

$$\sigma_0(f) = \Gamma_X \mid - \vdash \mathsf{get}\big(\mathsf{put}^{\pi_2(f\,0)}(\pi_1(f\,0)),\ \mathsf{put}^{\pi_2(f\,1)}(\pi_1(f\,1))\big)$$
$$\sigma_{n+1}(f) = \Gamma_X \mid a_1, \dots, a_{n+1} \vdash \mathsf{get}\big(\mathsf{close}(a_{n+1}; \sigma_n(f\,0)),\ \mathsf{close}(a_{n+1}; \sigma_n(f\,1))\big).$$

where we identify the elements of $X(0)$ with the variables in Γ_X. Like in the proof on Theorem 4.12, this definition of Γ_X works when $X(0)$ is finite; otherwise a finite Γ can be recovered from f.

We can show that σ is a homomorphism using Equations (27) to (33) from the parameterized theory. To show $\rho \circ \sigma = \mathsf{id}_{T_{\mathsf{l}}(X)}$, we proceed by induction on $n \in \mathbb{N}$ and use the fact that ρ is a homomorphism. For $\sigma \circ \rho = \mathsf{id}_{F_{\mathsf{l}}(X)}$, we use induction on terms and that both ρ and σ are homomorphisms. For the variable case, we use Equation (27). □

The interpretations in $T_{\mathsf{l}}(X)$ and $T'_{\mathsf{l}}(X)$, i.e., that of Piróg et al. [25], of programs with no open scopes agree.

Proposition 4.15. *Consider a fixed context of computation variables $\Gamma = (x_1 : 0, \dots, x_n : 0)$ and a truncated $X \in \mathbf{Set}^{\mathbb{N}}$. For any term $\Gamma \mid - \vdash t$, the following two interpretations coincide at index 0:*

$$[\![t]\!]_{T_{\mathsf{l}}(X),0} = [\![t]\!]_{T'_{\mathsf{l}}(X),0} : T_{\mathsf{l}}(X)(0)^n \to T_{\mathsf{l}}(X)(0),$$

under the identification $T_{\mathsf{l}}(X)(0) = T'_{\mathsf{l}}(X)(0)$.

Proof. The local operation binds parameters so we use a logical relation indexed by the length of the parameter context $p \in \mathbb{N}$. The relation has type

$$\mathcal{R}^p \subseteq \big(T_{\mathsf{l}}X(0)^n \to T_{\mathsf{l}}X(p)\big) \times \big(T_{\mathsf{l}}X(0)^n \to T'_{\mathsf{l}}X(p)\big)$$

and is defined as

$$\mathcal{R}^0 = \{(f, f) \mid f : T_{\mathsf{l}}X(0)^n \to T_{\mathsf{l}}X(0)\}$$
$$\mathcal{R}^{p+1} = \big\{(f, g) \mid \forall s \in 2.\ \big(\lambda(x_1, \dots, x_n).\, f(x_1, \dots, x_n)(s),\ \lambda(x_1, \dots, x_n).\, \pi_1 g(x_1, \dots, x_n)(s)\big) \in \mathcal{R}^p\big\},$$

where $(x_1, \dots, x_n) \in T_{\mathsf{l}}X(0)^n$, and π_1 is the projection $T_{\mathsf{l}}X(p) \times 2 \to T_{\mathsf{l}}X(p)$, c.f. (46).

The fundamental property for this logical relation says that for all terms $\Gamma \mid a_1, \dots, a_p \vdash t$, the two interpretations are related: $\big([\![t]\!]_{T_{\mathsf{l}}(X),0}, [\![t]\!]_{T'_{\mathsf{l}}(X),0}\big) \in \mathcal{R}^p$. If $p = 0$, this is enough to deduce the interpretations are equal. The fundamental property is proved by induction on t. □

The restriction of Γ in Proposition 4.15 to computation variables that do not depend on parameters and of Δ to be empty are reasonable because in the framework of Piróg et al. [25], only programs with no open scopes are well-formed. Therefore, only such programs can be substituted in t, justifying the restriction of $[\![t]\!]_{T'_{\mathsf{l}}(X)}$ to index 0.

4.4.4 Nondeterminism with cut. Recall the parameterized theory of explicit nondeterminism with cut from Example 3.9. According to Proposition 4.6, this theory has a free model for each $X \in \mathbf{Set}^{\mathbb{N}}$, denoted by $T_{\mathsf{cs}}(X)$.

Define a functor on sets Idem:

$$\mathsf{Idem}(A) = A \uplus \{a^* \mid a \in A\}.$$

Here in the notation a^*, the $*$ is just a flag. It is a semantic counterpart of cut, meaning "discard the yet uninspected choices and continue with a," following [26, Section 6]. (It is not the Kleene star of Proposition 2.14.)

If X is truncated, the carrier of the free model on X is

$$T_{\mathsf{cs}}(X)(n) = (\mathsf{Idem} \circ \mathsf{List})^{n+1}(X(0)).$$

(So these can be thought of as balanced binary trees, depth $n+1$, leaves labeled by $X(0)$, and where each non-leaf node is optionally flagged with $*$.)

When writing the interpretation of operations, we use the isomorphism

$$T_{\mathsf{cs}}(X)(0) \cong \mathsf{Idem}(\mathsf{List}(X(0))) \qquad T_{\mathsf{cs}}(X)(n+1) \cong \mathsf{Idem}(\mathsf{List}(T_{\mathsf{cs}}(X)(n)))$$

so an element of $T_{\mathsf{cs}}(X)(n)$ is either a list xs or a starred list xs^*.

The interpretation of cut and or in the free model is the following:

$$\mathsf{cut}_n : T_{\mathsf{cs}}(X)(n) \to T_{\mathsf{cs}}(X)(n) \quad \mathsf{cut}_n(\mathsf{xs}) = \mathsf{xs}^*, \quad \mathsf{cut}_n(\mathsf{xs}^*) = \mathsf{xs}^*$$

$$\mathsf{or}_n : T_{\mathsf{cs}}(X)(n) \times T_{\mathsf{cs}}(X)(n) \to T_{\mathsf{cs}}(X)(n)$$
$$\mathsf{or}_n(\mathsf{xs}, \mathsf{ys}) = \mathsf{xs} +\!\!+ \mathsf{ys}, \quad \mathsf{or}_n(\mathsf{xs}, \mathsf{ys}^*) = (\mathsf{xs} +\!\!+ \mathsf{ys})^*, \quad \mathsf{or}_n(\mathsf{xs}^*, -) = \mathsf{xs}^*.$$

The cut operation marks a list with a star; the second clause in the definition of cut corresponds to the idempotence Equation (36). Similarly to Section 4.4.1, or performs list concatenation, but now stars need to be taken into account as well. The second clause in definition of or corresponds to Equation (35), and the third clause to Equation (34).

The operation of opening a scope is interpreted as

$$\mathsf{scope}_n : T_{\mathsf{cs}}(X)(n+1) \to T_{\mathsf{cs}}(X)(n)$$
$$\mathsf{scope}_n(\mathsf{xs}^*) = \mathsf{scope}_n(\mathsf{xs}), \quad \mathsf{scope}_n([\,]) = [\,], \quad \mathsf{scope}_n(\mathsf{x} :: \mathsf{xs}) = \mathsf{or}_n(\mathsf{x}, \mathsf{scope}_n(\mathsf{xs})).$$

The first clause corresponds to erasing cuts as in Equation (38). The second clause corresponds to Equation (37). The third clause uses the interpretation of or to concatenate lists while taking into account cuts; it corresponds to Equation (39).

The close and fail operations have the same interpretation as in the nondeterminism with once example from Section 4.4.1:

$$\mathsf{close}_n : T_{\mathsf{cs}}(X)(n) \to T_{\mathsf{cs}}(X)(n+1) \qquad \mathsf{close}_n(x) = [x]$$
$$\mathsf{fail}_n : 1 \to T_{\mathsf{cs}}(X)(n) \qquad \mathsf{fail}_n() = [\,].$$

Theorem 4.16. *If $X \in \mathbf{Set}^{\mathbb{N}}$ is truncated, then $T_{\mathsf{cs}}(X)$ is the free model of the parameterized theory of nondeterminism with* cut*, defined in Example 3.9.*

Proof. We explained in this subsection why $T_{\mathsf{cs}}(X)$ together with its interpretation of operations respects the equations of the parameterized theory. Therefore, $T_{\mathsf{cs}}(X)$ is a model of the theory, in the sense of Definition 4.2.

Denote by $F_{\mathsf{cs}}(X)$ the free model on X for nondeterminism with cut, in the sense of Proposition 4.6. Consider the map $X \to T_{\mathsf{cs}}(X)$ that sends each element of $X(0)$ to the singleton list containing that element. By freeness (Definition 4.5), this map has a unique extension to a homomorphism of models $\rho : F_{\mathsf{cs}}(X) \to T_{\mathsf{cs}}(X)$. To show $T_{\mathsf{cs}}(X)$ is free, it is enough to show that ρ has an inverse.

Recall that $T_{\mathsf{cs}}(X)(n)$ is isomorphic to $(\mathsf{Idem} \circ \mathsf{List})^{n+1}(X(0))$, and $F_{\mathsf{cs}}(X)(n)$ contains equivalence classes of terms in context, where the computation variables come from X. We define a candidate inverse $\sigma : T_{\mathsf{cs}}(X) \to F_{\mathsf{cs}}(X)$ by induction on $n \in \mathbb{N}$ and on lists. For simplicity, we assume $X(0)$ is

finite and use representatives of equivalence classes from $F_{\mathbf{cs}}(X)(n)$ where the context Γ_X contains one variable $x_i : 0$ for each $c_i \in X(0)$, so we identify x_i and c_i in the definition below:

$$\begin{aligned}
\sigma_0(\mathsf{xs}^*) &= \Gamma_X \mid - \vdash \mathsf{cut}(\sigma_0(\mathsf{xs})) \quad \text{where } \mathsf{xs} \in \mathsf{List}(X(0)) \\
\sigma_0([\,]) &= \Gamma_X \mid - \vdash \mathsf{fail} \\
\sigma_0(\mathsf{x} :: \mathsf{xs}) &= \Gamma_X \mid - \vdash \mathsf{or}(\mathsf{x}, \sigma_0(\mathsf{xs}))
\end{aligned}$$

$$\begin{aligned}
\sigma_{n+1}(\mathsf{xs}^*) &= \Gamma_X \mid a_1, \dots, a_{n+1} \vdash \mathsf{cut}(\sigma_{n+1}(\mathsf{xs})) \\
\sigma_{n+1}([\,]) &= \Gamma_X \mid a_1, \dots, a_{n+1} \vdash \mathsf{fail} \\
\sigma_{n+1}(\mathsf{x} :: \mathsf{xs}) &= \Gamma_X \mid a_1, \dots, a_{n+1} \vdash \mathsf{or}\big(\mathsf{close}(a_{n+1}; \sigma_n(\mathsf{x})),\ \sigma_{n+1}(\mathsf{xs})\big).
\end{aligned}$$

If $X(0)$ is infinite, Γ_X can be defined to contain only the elements of $X(0)$ that appear in the term σ is constructing. The terms in the image of σ can be seen as the normal forms for the parameterized theory, defined recursively.

To show that σ is a section, $\rho \circ \sigma = \mathsf{id}_{T_{\mathbf{cs}}(X)}$, we use induction on $n \in \mathbb{N}$. For both the base case and the induction step, we first consider the case where the input is a list xs and use induction on lists. We use this intermediate result to prove the case of a starred list xs^*. Throughout, we use the fact that ρ is a homomorphism.

To show $\sigma \circ \rho = \mathsf{id}_{F_{\mathbf{cs}}}$, we fist show σ is a homomorphism of models, i.e., that it commutes with all the operations. The or and scope cases require induction on lists. Most importantly, we make use of equations from the parameterized theory of cut (Example 3.9). We then show by induction on the term formation rules that

$$\sigma_n\big(\rho_n(\Gamma_X \mid a_1, \dots, a_n \vdash t)\big) =_{\mathbf{cs}} \big(\Gamma_X \mid a_1, \dots, a_n \vdash t\big),$$

where the equality $=_{\mathbf{cs}}$ is in the theory of cut. We use the equations in the theory and the fact that σ is a homomorphism. The scope case also requires an induction on lists. □

4.5 A General Construction of a Parameterized Theory from Scoped Operations

The results of Section 4.4 can be understood as investigating particular instances of the following general problem. Given a monad $T : \mathbf{Set} \to \mathbf{Set}$ equipped with a set of scoped operations, find a parameterized algebraic theory $\mathcal{T}$ such that $\mathcal{T}$ induces, via the construction (44), the original monad T together with its associated scoped operations described in Proposition 4.8. In each example in Section 4.4, the given parameterized algebraic theories $\mathcal{T}$ have only finitely many equations, which together capture our computational intuitions for the scoped operations.

In this section we will give a solution to this problem in a more general case. Starting from any *finitary monad* $T : \mathbf{Set} \to \mathbf{Set}$ (i.e., those T which can be presented by an (ordinary) algebraic theory) and a set of scoped operations on T, we will find a parameterized algebraic theory $\mathcal{T}_{\mathbf{param}}$ which generates by Proposition 4.8 a monad isomorphic to T and for which the associated scoped operations agree (Theorem 4.17).

Unlike the examples considered in Section 4.4, $\mathcal{T}_{\mathbf{param}}$ will typically have infinitely many equations and there is no reason to expect the equational theory to fully capture the computational intuition behind the scoped operations. Indeed, in Proposition 4.19 we show that the $\mathcal{T}_{\mathbf{param}}$ construction does not fully recover the handcrafted parameterized theory of semi-determinism given in Figure 4. In particular, this shows that a given finitary monad equipped with scoped operations can be compatible with multiple inequivalent parameterized theories.

We now present the construction of $\mathcal{T}_{\mathbf{param}}$. Let $T : \mathbf{Set} \to \mathbf{Set}$ be a monad generated by an (ordinary) algebraic theory $\mathcal{T} = \langle \Sigma, E \rangle$, and let $\{\sigma_i : T^{k_i} \to T\}_{i \in I}$ be a set of scoped operations on T indexed by some set I, where $k_i \in \mathbb{N}$.

— The signature of $\mathcal{T}_{\mathbf{param}}$ is constructed by applying Example 3.3 to the signature Σ extended with scoped operation $\mathsf{sc}_i : (0 \mid 1, \dots, 1)$ for all $i \in I$, where the list $1, \dots, 1$ has length k_i.
— The equations of $\mathcal{T}_{\mathbf{param}}$ come in two kinds. First, each equation in E gives an equation in $\mathcal{T}_{\mathbf{param}}$ in the evident way. Second, for each $i \in I$ and $n \in \mathbb{N}$ and each k_i-tuple of Σ-terms $(x_1, \dots, x_n \vdash t_1, \dots, t_{k_i})$, the $\mathcal{T}$-equality class of those terms gives an argument tuple for $\sigma_{i,n} : T(n)^{k_i} \to T(n)$, and $\sigma_{i,n}([t_1], \dots, [t_{k_i}]) = [t']$ for some $x_1, \dots, x_n \vdash t'$. For each such argument tuple and each representative t' of the output, $\mathcal{T}_{\mathbf{param}}$ gets an equation:

$$x_1 : 0, \dots, x_n : 0 \mid - \vdash \mathsf{sc}_i(a.t_1[\mathsf{close}(a, x_i)/x_i]_{1 \le i \le n}, \dots, a.t_{k_i}[\mathsf{close}(a, x_i)/x_i]_{1 \le i \le n}) = t'. \tag{47}$$

One could slightly decrease the number of equations of the second kind in $\mathcal{T}_{\mathbf{param}}$ by making a single choice of representative t' of $\mathcal{T}$-equality class $\sigma_{i,n}([t_1], \dots, [t_{k_i}])$, since if t'' is another representative then $t' =_{\mathcal{T}_{\mathbf{param}}} t''$ is already a consequence just of equations of the first kind.

Theorem 4.17. *Let $\mathcal{T} = \langle \Sigma, E \rangle$ be an algebraic theory inducing a monad T on* **Set**, *equipped with scoped operations $\{\sigma_i\}_{i \in I}$ for some set I, each of arity k_i. With $\mathcal{T}_{\mathbf{param}}$ constructed as above, we have the following isomorphism:*

$$T \cong F'_{\mathcal{T}_{\mathbf{param}}} = \lfloor \circ F_{\mathcal{T}_{\mathbf{param}}} \circ \lceil .$$

Moreover, the scoped operations on $F'_{\mathcal{T}_{\mathbf{param}}}$, as constructed in Proposition 4.8, agree with the operations σ_i.

Proof. Note that $\mathcal{T}_{\mathbf{param}}$ extends the theory $\mathcal{T}_2$ considered in Proposition 4.9 by operations sc_i and equations of the form (47). In fact, we will show that the free model $F_{\mathcal{T}_{\mathbf{param}}}$ on a truncated object $\lceil A$, for a set A, is carried by $\bar{T}A \coloneqq \lambda m.\, T^{m+1}A$, the same as the free model of $\mathcal{T}_2$ from the proof of Proposition 4.9. We first note that for each $i \in I$, $\bar{T}A$ supports an interpretation of $\mathsf{sc}_i : (0 \mid 1, \dots, 1)$ with k_i arguments by the following morphism:

$$\big((\bar{T}A)(m+1)\big)^{k_i} = \big(T^{m+2}(A)\big)^{k_i} \xrightarrow{\sigma_{i,T^{m+1}(A)}} T^{m+2}(A) \xrightarrow{\mu_{T^m(A)}} T^{m+1}(A) = (\bar{T}A)(m+0).$$

We keep the same interpretation of $\mathsf{close} : (1 \mid 0)$, namely as the monadic unit

$$(\bar{T}A)(m+0) = T^{m+1}(A) \xrightarrow{\eta_{T^{m+1}A}} T^{m+2}(A) = (\bar{T}A)(m+1).$$

Since we saw in Proposition 4.9 that $\bar{T}A$ models $\mathcal{T}_2$, in order to check that $\bar{T}A$ is a model of $\mathcal{T}_{\mathbf{param}}$ we only need to see that all instances of (47) are validated. For terms $x_1 : 0, \dots, x_n : 0 \mid - \vdash t_j$ and each $1 \le j \le k_i$,

$$[\![t_j[\mathsf{close}(a, x_i)/x_i]_{1 \le i \le n}]\!]_m = (T^{m+1}A)^n \xrightarrow{([\![\mathsf{close}]\!]_m)^n = (\eta_{T^{m+1}A})^n} (T^{m+2}A)^n \xrightarrow{[\![t_j]\!]_{m+1}} T^{m+2}A.$$

The interpretation of the left-hand side of (47) is then

$$(T^{m+1}A)^n \xrightarrow{\langle [\![t_j]\!]_{m+1} \circ (\eta_{T^{m+1}A})^n \rangle_{1 \le j \le k_i}} (T^{m+2}A)^{k_i} \xrightarrow{\sigma_{i,T^{m+1}(A)}} T^{m+2}A \xrightarrow{\mu_{T^m(A)}} T^{m+1}A$$

whereas the right-hand side is

$$(T^{m+1}A)^n \xrightarrow{[\![t']\!]_m} T^{m+1}A.$$

Translating this into a statement about the ordinary algebraic theory $\mathcal{T}$ and **Set**-monad T, it suffices to show the following equation:

$$\begin{array}{c} (TB)^n \xrightarrow{\eta^n_{TB}} (TTB)^n \xrightarrow{\langle [\![t_j]\!]_{TB} \rangle_{1 \le j \le k_i}} (TTB)^{k_i} \xrightarrow{\sigma_{i,TB}} TTB \xrightarrow{\mu_B} TB \\ = \\ (TB)^n \xrightarrow{[\![t']\!]_B} TB \end{array} \tag{48}$$

for any set B.

For any *algebraic* operation $\mathrm{O} : T^n \to T$ supported by the monad T a short calculation shows that, for any set C and any function $\phi : n \to C$, we have an equation

$$1 \xrightarrow{\overline{\phi}} C^n \xrightarrow{\eta^n_C} (TC)^n \xrightarrow{\mathrm{O}_C} TC \quad = \quad 1 \xrightarrow{o} Tn \xrightarrow{T(\phi)} TC,$$

where the bar denotes currying and o is the generic effect associated with O, given by

$$1 \xrightarrow{o} Tn \quad = \quad 1 \xrightarrow{\overline{\eta_n}} (Tn)^n \xrightarrow{\mathrm{O}_n} Tn.$$

Hence, for any $\overline{\phi} : 1 \to (TB)^n$,

$$\begin{aligned}
& 1 \xrightarrow{\overline{\phi}} (TB)^n \xrightarrow{\eta^n_{TB}} (TTB)^n \xrightarrow{\langle [\![t_j]\!]_{TB} \rangle_{1 \le j \le k_i}} (TTB)^{k_i} \xrightarrow{\sigma_{i,TB}} TTB \xrightarrow{\mu_B} TB \\
= \quad & 1 \xrightarrow{\langle [t_j] \rangle_{1 \le j \le k_i}} (Tn)^{k_i} \xrightarrow{T(\phi)^k} (TTB)^{k_i} \xrightarrow{\sigma_{i,TB}} TTB \xrightarrow{\mu_B} TB \\
= \quad & 1 \xrightarrow{\langle [t_j] \rangle_{1 \le j \le k_i}} (Tn)^{k_i} \xrightarrow{\sigma_{i,n}} (Tn) \xrightarrow{T(\phi)} TTB \xrightarrow{\mu_B} TB \\
= \quad & 1 \xrightarrow{[t']} Tn \xrightarrow{T(\phi)} TTB \xrightarrow{\mu_B} TB \\
= \quad & 1 \xrightarrow{\overline{\phi}} (TB)^n \xrightarrow{\eta^n_{TB}} (TTB)^n \xrightarrow{[\![t']\!]_{TB}} TTB \xrightarrow{\mu_B} TB \\
= \quad & 1 \xrightarrow{\overline{\phi}} (TB)^n \xrightarrow{\eta^n_{TB}} (TTB)^n \xrightarrow{\mu^n_B} (TB)^n \xrightarrow{[\![t']\!]_B} TB \\
= \quad & 1 \xrightarrow{\overline{\phi}} (TB)^n \xrightarrow{[\![t']\!]_B} TB
\end{aligned}$$

and since $\overline{\phi}$ is arbitrary this proves Equation (48). This completes the proof that $\bar{T}A$ is a model of $\mathcal{T}_{\mathbf{param}}$.

Since $\bar{T}A$ is free on $\restriction A$ for the theory $\mathcal{T}_2$ of Proposition 4.9, which omits $\{\mathsf{sc}_i\}_{i \in I}$ and Equation (47), we only need to show that a $\mathcal{T}_2$-homomorphism $\bar{T}A \to Y$ into a $\mathcal{T}_{\mathbf{param}}$-model is always a $\mathcal{T}_{\mathbf{param}}$-homomorphism, i.e., that it commutes with every sc_i. But every element of $\bar{T}A(m+1) = T(\bar{T}A(m))$ is in the image of $[\![t]\!]_{m+1} \circ [\![\mathsf{close}]\!]_m$ for some Σ-term $x_1, \ldots, x_n \vdash t$, thus this amounts to the hypothesis that Y validates all instances of Equation (47). □

Remark 4.18. Whereas in Section 3.2 we constructed scoped theories out of particular algebraic theories whose monads support a scoped operation, the recipe for $\mathcal{T}_{\mathbf{param}}$ constructs a scoped theory in the general case. However, the presentation of $\mathcal{T}_{\mathbf{param}}$ typically includes infinitely many equations whereas the theories in Section 3.2 are given finite presentations. For example, in the case of once (Figure 4), Equations (21) and (23) are not part of $\mathcal{T}_{\mathbf{param}}$ but infinitely many substitution instances of them are. The free models of $\mathcal{T}_{\mathbf{param}}$ and that in Section 4.4.1 agree on truncated X (since the free model of the algebraic theory of explicit nondeterminism is List), which shows that the theory of once admits derivations of all equations of the corresponding $\mathcal{T}_{\mathbf{param}}$. However, as we show below, Equations (21) and (23) are not derivable in $\mathcal{T}_{\mathbf{param}}$. The issue is that (47) only governs the behavior of scoped operations when applied to arguments definable using the algebraic operations of $\mathcal{T}$. We argue that the full behavior of once is captured by the theory in Figure 4, rather than by $\mathcal{T}_{\mathbf{param}}$.

Proposition 4.19. *Equations* (21) *and* (23) *are not derivable in the* $\mathcal{T}_{\mathbf{param}}$ *of the monad* $\mathsf{List} : \mathbf{Set} \to \mathbf{Set}$ *equipped with the scoped operation* $\mathsf{once} : \mathsf{List} \to \mathsf{List}$ *from Example 2.13, where* $\mathsf{once}_X([]) = []$ *and* $\mathsf{once}_X([x, \ldots]) = [x]$.

Proof notes. We give a model of $\mathcal{T}_{\mathbf{param}}$ in which the equations do not hold, by taking a simple variation of the free $\mathcal{T}_{\mathbf{param}}$-model. For any fixed set X, define $A \in \mathbf{Set}^{\mathbb{N}}$ by

$$A(0) = \mathsf{List}(X + \{\star_0\}) \qquad A(n+1) = \mathsf{List}(A(n) + \{\star_{n+1}\}).$$

This is a levelwise model of explicit nondeterminism in the obvious way. For the rest of the structure we define $\mathsf{close}_{A,n} : A(n) \to A(n+1)$ by $\mathsf{close}_{A,n}(x) = [x]$ and $\mathsf{once}_n : A(n+1) \to A(n)$ by

$$\mathsf{once}_{A,n}([]) = [], \qquad \mathsf{once}_{A,n}([x_0, x_1, \ldots, x_m]) = x_0 \text{ if all } x_i \in A(n),$$

$$\mathsf{once}_{A,n}([x_0, x_1, \ldots, x_m]) = c_n(x_0) +\!\!+ \ldots +\!\!+ c_n(x_m) \text{ if some } x_i = \star_{n+1},$$

where $c_n(x) = x$ if $x \in A(n)$ and $c_n(\star_{n+1}) = [\star_n]$. Now each instance of Equation (47) holds in A, roughly because in the left-hand side $\mathsf{once}_{A,n}$ only gets applied to a list of elements in the image of $\mathsf{close}_{A,n}$ (so no $\star_{n+1}$'s). To see that (21) and (23) do not hold in A, observe the following:

$$[\![\mathsf{once}(a.\mathsf{or}(x(a), x(a)))]\!]_{A,0}([\star_1]) = [\star_0, \star_0] \neq [\star_0] = [\![\mathsf{once}(a.x(a))]\!]_{A,0}([\star_1])$$
$$[\![\mathsf{once}(a.\mathsf{or}(\mathsf{close}(a; x), y(a)))]\!]_{A,0}([\star_0], [\star_1]) = [\star_0, \star_0] \neq [\star_0] = [\![x]\!]_{A,0}([\star_0], [\star_1]). \qquad \square$$

5 Summary and Research Directions

We have provided a fresh perspective on scoped effects in terms of the formalism of parameterized algebraic theories, using the idea that scopes are resources (Example 3.3). As parameterized algebraic theories have a sound and complete algebraic theory (Propositions. 4.3, and 4.6), this carries over to a sound and complete equational theory for scoped effects. We showed that our approach recovers the earlier models for scoped nondeterminism, exceptions, and state (Theorems 4.12–4.14, and 4.16).

Here we have focused on equational theories for effects alone. But as is standard with algebraic effects, it is easy to add function types, inductive types, and so on, together with standard beta/eta theories (e.g., [30], [40, Section 5]). This can be shown sound by the simple models considered here, as indeed the canonical model $\mathbf{Set}^{\mathbb{N}}$ is closed and has limits and colimits.

Combinations of Theories. A more concrete research direction is to define combinations of parameterized theories, and hence of scoped effects, using sum and tensor, generalizing combinations of algebraic effects [13]. Informally, the tensor corresponds to all the operations from one theory commuting with the operations from the other. But in the case of parameterized theories in $\mathbf{Set}^{\mathbb{N}}$, defining what it means for operations to commute is delicate because the substitution rule (16) is more complex than in the algebraic case (but see [38, Definition 2]).

For example, consider the commuting combination of explicit nondeterminism with once (Figure 4) and global state (Example 2.8). In the resulting theory, the state is rolled back when a computation fails. For example, the fact that put^0 commutes with fail means we can prove:

$$\begin{aligned}\mathsf{put}^1\big(\mathsf{or}(\mathsf{put}^0(\mathsf{fail}),\ \mathsf{get}(x_0, x_1))\big) &= \mathsf{put}^1\big(\mathsf{or}(\mathsf{fail},\ \mathsf{get}(x_0, x_1))\big)\\ &= \mathsf{put}^1\big(\mathsf{get}(x_0, x_1)\big)\\ &= \mathsf{put}^1(x_1)\end{aligned}$$

using commutativity of put^0 and fail for the first step. The fact that put and get commute with once and close means the state operations are independent of scopes.

Our work opens up new directions for scoped effects, in theory and in practice. By varying the substructural laws of parameterized algebraic theories, we can recover foundations for scoped effects where scopes (as resources) can be reordered or discarded, i.e., where they are not well-bracketed, already considered briefly in the literature [25]. For example, the parameterized algebraic theory of qubits [40] might be regarded as a scoped effect, where we open a scope when a qubit is allocated and close the scope when it is discarded; this generalizes traditional scoped effects as multi-qubit operations affect multiple scopes.

Acknowledgments

We are grateful to many colleagues for helpful discussions and to the ESOP 2024 reviewers for their helpful comments and suggestions.

References

[1] Samson Abramsky and Achim Jung. 1994. Domain theory. In *Handbook of Logic in Computer Science*. S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum (Eds.), Clarendon Press, Vol. 3.

[2] Andrej Bauer. 2019. What is algebraic about algebraic effects and handlers? arxiv:1807.05923. Retrieved from https://doi.org/10.48550/arXiv.1807.05923

[3] Nick Benton and Philip Wadler. 1996. Linear logic, monads and the lambda calculus. In *11th Annual IEEE Symposium on Logic in Computer Science*, 420–431. DOI: https://doi.org/10.1109/LICS.1996.561458

[4] Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus E. Møgelberg, and Lars Birkedal. 2016. Guarded dependent type theory with coinductive types. In *Foundations of Software Science and Computation Structures*. Bart Jacobs and Christof Löding (Eds.), Lecture Notes in Computer Science, Vol. 9634, Springer, Berlin, 20–35. DOI: https://doi.org/10.1007/978-3-662-49630-5_2

[5] Roger Bosman, Birthe van den Berg, Wenhao Tang, and Tom Schrijvers. 2023. A calculus for scoped effects & handlers. arxiv:2304.09697. Retrieved from https://doi.org/10.48550/arXiv.2304.09697

[6] 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, 1–38.

[7] Marcelo Fiore and Dmitrij Szamozvancev. 2022. Formal metatheory of second-order abstract syntax. *Proceedings of the ACM on Programming Languages* 6, POPL (2022), 1–29. DOI: https://doi.org/10.1145/3498715

[8] Marcelo P. Fiore and Chung-Kil Hur. 2010. Second-order equational logic (extended abstract). In *24th International Workshop on Computer Science Logic (CSL '10), 19th Annual Conference of the EACSL*. Anuj Dawar and Helmut Veith (Eds.), Lecture Notes in Computer Science, Vol. 6247, Springer, 320–335. DOI: https://doi.org/10.1007/978-3-642-15205-4_26

[9] Marcelo P. Fiore and Ola Mahmoud. 2010. Second-order algebraic theories—(extended abstract). In *35th International Symposium on Mathematical Foundations of Computer Science (MFCS '10)*. Petr Hlinený and Antonín Kucera (Eds.), Lecture Notes in Computer Science, Vol. 6281, Springer, 368–380. DOI: https://doi.org/10.1007/978-3-642-15155-2_33

[10] Marcelo P. Fiore and Sam Staton. 2014. Substitution, jumps, and algebraic effects. In *the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (CSL-LICS '14)*.

[11] Neil Ghani, Tarmo Uustalu, and Makoto Hamana. 2006. Explicit substitutions and higher-order syntax. *Higher-Order and Symbolic Computation* 19, 2–3 (2006), 263–282. DOI: https://doi.org/10.1007/s10990-006-8748-4

[12] Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science* 50 (1987), 1–102.

[13] Martin Hyland, Gordon D. Plotkin, and John Power. 2006. Combining effects: sum and tensor. *Theoretical Computer Science* 357, 1 (July 2006), 70–99. DOI: https://doi.org/10.1016/j.tcs.2006.03.013

[14] Shin-ya Katsumata, Dylan McDermott, Tarmo Uustalu, and Nicolas Wu. 2022. Flexible presentations of graded monads. *Proceedings of the ACM on Programming Languages* 6, ICFP, Article 123 (Aug. 2022), 29 pages. DOI: https://doi.org/10.1145/3547654

[15] 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. DOI: https://doi.org/10.1016/0022-4049(93)90092-8

[16] F. William Lawvere. 1963. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences* 50, 5 (1963), 869–872. DOI: https://doi.org/10.1073/pnas.50.5.869

[17] Sam Lindley, Cristina Matache, Sean K. Moss, Sam Staton, Nicolas Wu, and Zhixuan Yang. 2024. Scoped effects as parameterized algebraic theories. In *33rd European Symposium on Programming on Programming Languages and Systems (ESOP '24)*.

[18] F. E. J. Linton. 1966. Some aspects of equational categories. In *the Conference on Categorical Algebra*. S. Eilenberg, D. K. Harrison, S. MacLane, and H. Röhrl (Eds.), Springer, Berlin, 84–94. DOI: https://doi.org/10.1007/978-3-642-99902-4_3

[19] Paul-André Melliès. 2014. Local states in string diagrams. In *Rewriting and Typed Lambda Calculi*. Gilles Dowek (Ed.), Springer International Publishing, Cham, 334–348.

[20] Paul-André Melliès. 2010. Segal condition meets computational effects. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, 150–159. DOI: https://doi.org/10.1109/LICS.2010.46

[21] Robin Milner. 1999. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press.

[22] Eugenio Moggi. 1989. Computational lambda-calculus and monads. In *4th Annual Symposium on Logic in Computer Science*, 14–23. DOI: https://doi.org/10.1109/LICS.1989.39155

[23] Eugenio Moggi. 1991. Notions of computation and monads. *Information and Computation* 93, 1 (1991), 55–92. DOI: https://doi.org/10.1016/0890-5401(91)90052-4
[24] Leaf Petersen, Robert Harper, Karl Crary, and Frank Pfenning. 2003. A type theory for memory allocation and data layout. In *the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '03)*.
[25] Maciej Piróg, Tom Schrijvers, Nicolas Wu, and Mauro Jaskelioff. 2018. Syntax and semantics for operations with scopes. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '18)*. ACM, New York, NY, 809–818. DOI: https://doi.org/10.1145/3209108.3209166
[26] Maciej Piróg and Sam Staton. 2017. Backtracking with cut via a distributive law and left-zero monoids. *Journal of Functional Programming* 27 (2017), e17.
[27] Andrew M. Pitts. 2011. Structural recursion with locally scoped names. *Journal of Functional Programming* 21, 3 (2011), 235–286. DOI: https://doi.org/10.1017/S0956796811000116
[28] Gordon D. Plotkin and John Power. 2001. Adequacy for algebraic effects. In *International Conference on Foundations of Software Science and Computation Structures (FOSSACS '01)*. Furio Honsell and Marino Miculan (Eds.), Lecture Notes in Computer Science, Vol. 2030, Springer, 1–24. DOI: https://doi.org/10.1007/3-540-45315-6_1
[29] Gordon D. Plotkin and John Power. 2002. Notions of computation determine monads. In *5th International Conference on Foundations of Software Science and Computation Structures (FOSSACS '02)*. Mogens Nielsen and Uffe Engberg (Eds.), Springer, 342–356. DOI: https://doi.org/10.1007/3-540-45931-6_24
[30] Gordon D. Plotkin and John Power. 2003. Algebraic operations and generic effects. *Applied Categorical Structures* 11 (2003), 69–94.
[31] Gordon D. Plotkin and John Power. 2004. Computational effects and operations: An overview. *Electronic Notes in Theoretical Computer Science 73* (Oct. 2004), 149–163. DOI: https://doi.org/10.1016/j.entcs.2004.08.008
[32] Gordon D. Plotkin and Matija Pretnar. 2013. Handling algebraic effects. *Logical Methods in Computer Science* 9, 4:23 (Dec. 2013), 1–36 DOI: https://doi.org/10.2168/lmcs-9(4:23)2013
[33] Jeff Polakow. 2001. *Ordered Linear Logic and Applications*. Ph.D. Dissertation.
[34] John Power. 1999. Enriched Lawvere theories. *Theory and Applications of Categories* 6, 7 (1999), 83–93.
[35] John Power. 2006. Semantics for local computational effects. *Electronic Notes in Theoretical Computer Science* 158 (May 2006), 355–371. DOI: https://doi.org/10.1016/j.entcs.2006.04.018
[36] Ian Stark. 2008. Free-algebra models for the pi -calculus. *Theoretical Computer Science* 390, 2–3 (2008), 248–270. DOI: https://doi.org/10.1016/j.tcs.2007.09.024
[37] Sam Staton. 2013. An algebraic presentation of predicate logic - (extended abstract). In *International Conference on Foundations of Software Science and Computational Structures (FOSSACS '13)*.
[38] Sam Staton. 2013. Instances of computational effects: An algebraic perspective. In *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '13)*.
[39] Sam Staton. 2014. Freyd categories are enriched lawvere theories. *Electronic Notes in Theoretical Computer Science* 303 (2014), 197–206. DOI: https://doi.org/10.1016/j.entcs.2014.02.010
[40] Sam Staton. 2015. Algebraic effects, linearity, and quantum programming languages. In *42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15)*.
[41] S. Staton, D. Stein, H. Yang, N. L. Ackerman, C. Freer, and D. Roy. 2018. The Beta-Bernoulli process and algebraic effects. In *45th International Colloquium on Automata, Languages, and Programming (ICALP '18)*.
[42] Patrick Thomson, Rob Rix, Nicolas Wu, and Tom Schrijvers. 2022. Fusing industry and academia at GitHub (experience report). *Proceedings of the ACM on Programming Languages* 6, ICFP, Article 108 (Aug. 2022), 16 pages. DOI: https://doi.org/10.1145/3547639
[43] Birthe van den Berg and Tom Schrijvers. 2023. A framework for higher-order effects & handlers. arXiv:2302.01415 Retrieved from https://doi.org/10.48550/arXiv.2302.01415
[44] Nicolas Wu, Tom Schrijvers, and Ralf Hinze. 2014. Effect handlers in scope. *2014 ACM SIGPLAN Symposium on Haskell (Haskell '14)*, 1–12. DOI: https://doi.org/10.1145/2633357.2633358
[45] Zhixuan Yang, Marco Paviotti, Nicolas Wu, Birthe van den Berg, and Tom Schrijvers. 2022. Structured handling of scoped effects. Springer-Verlag, Berlin, 462–491. DOI: https://doi.org/10.1007/978-3-030-99336-8_17
[46] Zhixuan Yang and Nicolas Wu. 2023. Modular models of monoids with operations. *Proceedings of the ACM on Programming Languages* 7, ICFP, Article 208 (Aug. 2023), 38 pages. DOI: https://doi.org/10.1145/3607850

Received 27 April 2024; revised 29 December 2024; accepted 4 March 2025