UNIVERSITY OF BIRMINGHAM

University of Birmingham Research at Birmingham

Scoped Effects as Parameterized Algebraic Theories

Lindley, Sam; Matache, Cristina; Moss, Sean; Staton, Sam; Wu, Nicolas; Yang, Zhixuan

DOI:
10.1007/978-3-031-57262-3_1

License:
Creative Commons: Attribution (CC BY)

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Lindley, S, Matache, C, Moss, S, Staton, S, Wu, N & Yang, Z 2024, Scoped Effects as Parameterized Algebraic Theories. in S Weirich (ed.), Programming Languages and Systems: 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6–11, 2024, Proceedings, Part I. 1 edn, Lecture Notes in Computer Science, vol. 14576, Springer, pp. 3-21, ESOP 2024, Luxembourg City, Luxembourg, 8/04/24. https://doi.org/10.1007/978-3-031-57262-3_1

Link to publication on Research at Birmingham portal

General rights
Unless a licence is specified above, all rights (including copyright and moral rights) in this document are retained by the authors and/or the copyright holders. The express permission of the copyright holder must be obtained for any use of this material other than for purposes permitted by law.

•Users may freely distribute the URL that is used to identify this publication.
•Users may download and/or print one copy of the publication from the University of Birmingham research portal for the purpose of private study or non-commercial research.
•User may use extracts from the document in line with the concept of ‘fair dealing’ under the Copyright, Designs and Patents Act 1988 (?)
•Users may not further distribute the material nor use it for the purposes of commercial gain.

Where a licence is displayed above, please note the terms and conditions of the licence govern your use of this document.

When citing, please reference the published version.

Take down policy
While the University of Birmingham exercises care and attention in making items available there are rare occasions when an item has been uploaded in error or has been deemed to be commercially or otherwise sensitive.

If you believe that this is the case for this document, please contact UBIRA@lists.bham.ac.uk providing details and we will remove access to the work immediately and investigate.

Download date: 04. May. 2024

Scoped Effects as Parameterized Algebraic Theories

Sam Lindley[1], Cristina Matache[1(✉)], Sean Moss[2], Sam Staton[3], Nicolas Wu[4], and Zhixuan Yang[4]

[1] University of Edinburgh, Edinburgh, UK
{sam.lindley,cristina.matache}@ed.ac.uk
[2] University of Birmingham, Birmingham, UK
s.k.moss@bham.ac.uk
[3] University of Oxford, Oxford, UK
sam.staton@cs.ox.ac.uk
[4] Imperial College London, London, UK
{n.wu,s.yang20}@imperial.ac.uk

Abstract. Notions of computation can be modelled 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 fresh perspective by way of equational characterizations of several known models of scoped effects.

Keywords: algebraic effects · scoped effects · monads · category theory · algebraic theories.

1 Introduction

The central idea of *algebraic effects* [29] is that impure computation can be built and reasoned about equationally, using an algebraic theory. *Effect handlers* [28] 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 paper 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* (Sec. 2.2) is that certain parts of an impure computation should be dealt with one way, and other parts another way,

© The Author(s) 2024
S. Weirich (Ed.): ESOP 2024, LNCS 14576, pp. 3–21, 2024.
https://doi.org/10.1007/978-3-031-57262-3_1

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 [41,23,42,5,40,43] 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 [41] and timing analysis in telemetry [39].

Parameterized algebraic theories (Sec. 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 [35], channels in the π-calculus [36], code pointers [7], qubits in quantum programming [38], and urns in probabilistic programming [34].

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 fresh perspective provides:

- the first syntactic sound and complete equational reasoning system for scoped effects, based on the equational reasoning for parameterized algebraic theories (Prop. 2, Prop. 3);
- a canonical notion of semantic model for scoped effects supporting three key examples from the literature: nondeterminism with semi-determinism (Thm. 2), catching exceptions (Thm. 3), and local state (Thm. 4); and
- a reconstruction of the previous categorical analysis of scoped effects via the categorical analysis of 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 (Thm. 1).

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 1 and 4 for further details.) Let us begin with two algebraic operations: $\mathsf{or}(x, y)$, which nondeterministically chooses between continuing[5] 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

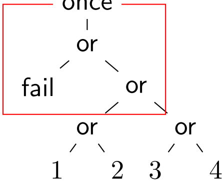

Fig. 1. Illustrating (1)

[5] This continuation-passing style is natural for algebraic effects, but when programming one often uses equivalent direct-style *generic effects* [25] such as $\underline{\mathsf{or}} : \mathsf{unit} \to \mathsf{bool}$, where $\mathsf{or}(x, y)$ can be recovered by pattern matching on the result of $\underline{\mathsf{or}}$.

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.

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

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 Fig. 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 (§3) and the equations for nondeterminism (Fig. 2), we can prove that the term (2) is equivalent to $\mathsf{or}(1,2)$.

2 Background

2.1 Algebraic effects

Moggi [20,21] shows that many non-pure features of programming languages, typically referred to as *computational effects*, can be modelled uniformly as *monads*, but the question is — *how do we construct a monad for an effect*, or putting it differently, *where do the monads modelling effects come from*? A classical result in category theory is that finitary monads over the category of sets are equivalent to *algebraic theories* [16,15]: 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 [26] show that many monads that are used for modelling 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 [26], which is nowadays commonly referred to as *algebraic effects*:

Perspective 1 ([26]). 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 in a simple form here; see [27,2] for more discussion.

Definition 1. *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 an* arity *to each operation.*

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 the following rules:

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

As usual we will consider terms up to renaming of variables. Thus a context $\Gamma = (x_1, \ldots, x_n)$ can be identified with the natural number n, and Tm_Σ can be thought of as a function $\mathbb{N} \to \mathbf{Set}$.

Example 1. 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

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

Example 2. The signature of *mutable state* of a single bit has operations:

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

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 [26].

Definition 2. *A (first-order finitary)* algebraic theory $T = \langle \Sigma, E \rangle$ *is a signature* Σ *(Def. 1) 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 3. 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 4. The theory of *explicit nondeterminism* has the signature in Example 1 and the following equations saying that fail and or form a monoid:

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

Example 5. The theory of *mutable state* has the signature in Example 2 and the following equations for all $i, i' \in \{0, 1\}$:

$$x_0, x_1 \vdash \mathsf{put}^i(\mathsf{get}(x_0, x_1)) = \mathsf{put}^i(x_i) \qquad x \vdash \mathsf{put}^i(\mathsf{put}^{i'}(x)) = \mathsf{put}^{i'}(x)$$

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

Every algebraic theory gives rise to a monad by the *free-algebra construction*, which we will discuss in a more general setting in Section 3. The three examples above respectively give rise to the monads $(1 + -)$, List, $(- \times 2)^2$ on the category of sets that are used to give semantics to the respective computational effects in programming languages [20,21]. In this way, the monad for a computational effect

is constructed in a very 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 characterise the interaction between the two theories [12]. By contrast, monads are not composable in general.

The kind of plain algebraic theory encapsulated by Def. 2 above is not, however, sufficiently expressive enough for some programming language applications. In this paper we focus on two problems with plain algebraic theories:

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

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

   On the other hand, 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 $\mathsf{openFile}$ 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 generalising Def. 2 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 fresh perspective of this paper is that scoped effects can be fruitfully understood as a *non-commutative linear* variant of parameterized effects.

2.2 Scoped effects

Recall that our first problem with plain algebraic theories 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 operations such as opening a file in a scope, running a program concurrently in a scope, and looping a program in a scope.

Operations delimiting scopes are treated as *handlers* (i.e. models) of algebraic operations by Plotkin and Pretnar [28], instead of operations in their own right. The following alternative perspective was first advocated by Wu et al. [41].

Perspective 2 ([41]). Scoped operations are *operations that do not commute with substitution*, since sequential composition in monads generated from algebraic theories corresponds to *substitution*. Such operations arise in contexts other than computational effects as well, for example, the later modality in *guarded dependent type theory* (GDTT) [4].

Extensions of algebraic effects to accommodate scoped operations were first studied by Wu et al. [41] 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, the program $s(\mathsf{put}^0); \mathsf{put}^1(x)$, where put^0 is wrapped in the scope of s, is encoded formally as
$$\mathsf{begin}_s(\mathsf{put}^0(\mathsf{end}_s(\mathsf{put}^1(x)))).$$
2. The *higher-order abstract syntax (HOAS) approach* directly constructs a monad for programs with algebraic and scoped operations. In Haskell, their monad for programs with algebraic operations parameterized by a signature functor `asig` and scoped operations parameterized by a functor `ssig` is
```
data Prog a where
  Ret :: a -> Prog a
  Alg :: asig (Prog a) -> Prog a
  Scp :: forall x. ssig (Prog x) -> (x -> Prog a) -> Prog a
```
where `Scp p f` represents a scoped operation acting on a program `p` followed by a program `f` after the scope (cf *delayed substitution* in GDTT [4]).

The HOAS 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 [23,43,40,42] and operational semantics for scoped effects has also been developed [5]. Of particular relevance to the current paper is the work of Piróg et al. [23], which we briefly review in the rest of this section.

Piróg et al. [23] 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. 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:
$$(\triangleright X)(0) = \emptyset, \qquad (\triangleright X)(n+1) = X(n), \qquad (\triangleleft X)(n) = X(n+1). \tag{3}$$

These two functors are closely related to bracketing: 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.

Given two signatures Σ and Σ' as in Def. 1 for algebraic and scoped operations respectively, let $\bar{\Sigma}, \bar{\Sigma}' : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set}^{\mathbb{N}}$ be the functors given by

$$(\bar{\Sigma}X)(n) = \textstyle\coprod_{o \in |\Sigma|} X(n)^{ar(o)} \quad \text{and} \quad (\bar{\Sigma}'X)(n) = \textstyle\coprod_{s \in |\Sigma'|} X(n)^{ar(s)}.$$

Moreover, for every $A \in \mathbf{Set}$, let $\upharpoonleft A \in \mathbf{Set}^{\mathbb{N}}$ be given by

$$(\upharpoonleft A)(0) = A \qquad\qquad (\upharpoonleft A)(n+1) = 0,$$

and conversely for every $X \in \mathbf{Set}^{\mathbb{N}}$, let $\downharpoonleft X \in \mathbf{Set}$ be given by $\downharpoonleft X = X(0)$.

Proposition 1 (Piróg et al. [23]). *The following functor can be extended to a monad that is isomorphic to the monad* Prog *in the HOAS approach above:*

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

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

The monad from Prop. 1 is a way of specifying the syntax of programs with algebraic and scoped operations, without taking into account equations. In [23], a *model* of a scoped effect is an *algebra* for the monad $\left(\bar{\Sigma} + \left(\bar{\Sigma}' \circ \triangleleft\right) + \triangleright\right)^{*}$. In Thms. 2–4, we show that three examples of models from [23] are *free algebras* on $\upharpoonleft A \in \mathbf{Set}^{\mathbb{N}}$ 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 [32] in a purely categorical setting. A syntactic framework extending that of algebraic theories, called *parameterized algebraic theories*, was introduced by Staton [35,36] and is used to give an axiomatic account of local computational effects such as restriction [24], local state [26], and the π-calculus [19,33].

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 paper, we propose to interpret parameters as *names of scopes*.

Perspective 3. 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 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 [36, §V.E]. A closed term can only mention locations introduced by new^i, meaning that 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 §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 §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 (Thms. 2–4).

Our parameters are linear in the same sense as variables in linear logic and linear lambda calculi e.g. [11,3], but with an additional non-commutativity restriction. Non-commutative linear systems are also known as ordered linear systems e.g. [30,22]. A commutative linear version of parameterized algebraic theories was considered in [38] to give an algebraic theory of quantum computation; in this case, parameters stand for qubits.

Remark 1. Parameterized algebraic theories characterize a certain class of enriched monads [35], extending the correspondence between algebraic theories and monads on the category of sets, and the idea of Plotkin and Power [26] that computational effects give rise to monads (see §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 [13], which also use presheaf-enrichment; second-order algebra [8,9,10], which also use variable binding; and graphical methods [17], which also connect to presheaf categories.

3 Parameterized theories of scoped effects

In order to describe scoped effects we use a substructural version of parameterized algebraic theories [35]. A theory consists of a signature (Def. 3) and

$$x : 0, y : 0, z : 0 \mid - \vdash \mathsf{or}(\mathsf{or}(x, y), z) = \mathsf{or}(x, \mathsf{or}(y, z)) \quad (4)$$

$$x : 0 \mid - \vdash \mathsf{or}(x, \mathsf{fail}) = x \qquad x : 0 \mid - \vdash \mathsf{or}(\mathsf{fail}, x) = x \quad (5)$$

$$- \mid - \vdash \mathsf{once}(a.\mathsf{fail}) = \mathsf{fail} \qquad x{:}1 \mid - \vdash \mathsf{once}(a.\mathsf{or}(x(a),\ x(a))) = \mathsf{once}(a.x(a)) \quad (6)$$

$$x{:}0 \mid - \vdash \mathsf{once}(a.\mathsf{close}(a, x)) = x \qquad x{:}0, y{:}1 \mid - \vdash \mathsf{once}\big(a.\mathsf{or}(\mathsf{close}(a, x),\ y(a))\big) = x \quad (7)$$

Fig. 2. The parameterized theory of explicit nondeterminism (4–5) and once (6–7). Terms-in-context are defined further down.

equations (Def. 4) between terms formed from the signature. 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.

Definition 3. *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 $ar(\mathrm{O}) = (p \mid m_1 \dots m_k)$ *consisting of a natural number* p *and a list of natural numbers* $m_1, \dots, m_k$*. This means that the operation* O *takes in* p *parameters and* k *continuations, and it binds* m_i *parameters in the* i*-th continuation.*

Remark 2. Given signatures for algebraic and scoped operations, as in Def. 1 and §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 $(0 \mid 0 \dots 0)$, where the list $0 \dots 0$ has length k;
- for each scoped operation $(\mathsf{sc} : k)$ of arity $k \in \mathbb{N}$, there is a parameterized operation $\mathsf{sc} : (0 \mid 1 \dots 1)$, where the list $1 \dots 1$ 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.

Example 6. The algebraic theory of explicit nondeterminism in Example 1 can be extended with a *semi-determinism* operator once:

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

The continuation of once opens a new scope, by binding a parameter. Inside this scope, only the first successful branch of or is kept. The term formation rules below 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. See Fig. 2 for equations.

For a given signature, we define the terms-in-context of *algebra with non-commutative linear parameters*. 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

$a_1, \ldots, a_m$. Terms $\Gamma \mid \Delta \vdash t$ are inductively generated by the following two rules.

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

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

In the conclusion of the last rule, the parameters $a_1 \dots a_p$ are consumed by the operation O. The parameters $b_1 \dots b_{m_i}$ are bound in t_i. As usual, we treat all terms up to renaming of variables.

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 in the opposite order that they were opened, that is, scopes are well-bracketed. 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.

Compared to the *algebra with linear parameters* of [38], used for describing quantum computation, 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 more clear the connection to non-linear parameterized theories [35,36].

The syntax admits the following simultaneous substitution rule:

$$\frac{\begin{array}{c}(x_1 : m_1 \dots x_l : m_l) \mid \Delta \vdash t \\ \Gamma' \mid \Delta', a_1 \dots a_{m_1} \vdash t_1 \qquad \cdots \qquad \Gamma' \mid \Delta', a_1 \dots a_{m_l} \vdash t_l\end{array}}{\Gamma' \mid \Delta', \Delta \vdash t\big[(\Delta', a_1 \dots a_{m_1} \vdash t_1)/x_1 \ \dots\ (\Delta', a_1 \dots a_{m_l} \vdash t_l)/x_l\big]} \tag{8}$$

In the conclusion, the notation $(\Delta', a_1 \dots a_{m_i} \vdash t_i)/x_i$ emphasizes that the parameters $(a_1 \dots a_{m_i})$ 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.

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 (8), 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 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 in the same context* $(\Gamma \mid \Delta)$ *for some* Γ *and* Δ.

We will omit the qualifier "with non-commutative linear parameters" where convenient and refer to "parameterized theories" or just "theories". Given a theory $\mathcal{T}$, we form a system of equivalence relations $=_{\mathcal{T},(\Gamma\mid\Delta)}$ on terms in each context $(\Gamma \mid \Delta)$ by closing substitution instances of the axioms under reflexivity, symmetry, transitivity, and congruence.

Example 7. As we mentioned earlier, *exception catching* is not an ordinary algebraic operation. As parameterized operations, 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 $\mathsf{catch}(a.\mathsf{close}(a, x), b.y(b))$, exceptions in x will not be caught, because the scope of the catch has already been closed. The equations are:

$$y{:}0 \mid - \vdash \mathsf{catch}(a.\mathsf{throw},\ b.\mathsf{close}(b, y)) = y \tag{9}$$

$$- \mid - \vdash \mathsf{catch}(a.\mathsf{throw},\ b.\mathsf{throw}) = \mathsf{throw} \tag{10}$$

$$x{:}0, y{:}1 \mid - \vdash \mathsf{catch}\big(a.\mathsf{close}(a, x),\ b.y(b)\big) = x \tag{11}$$

Remark 3. The arity of catch from Ex. 7 corresponds to the signature used in [23, Ex. 4.5]. Using the extra flexibility of parameterized algebraic theories, we could instead consider the arity $\mathsf{catch} : (0 \mid 1, 0)$. This seems more natural as there is no need to delimit a scope in the second continuation, which handles the exceptions.

Example 8 (Mutable state with local values). The theory of (boolean) mutable state with one memory location (Ex. 2) 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 [26,18]:

$$z : 0 \mid - \vdash \mathsf{get}(\mathsf{put}^0(z), \mathsf{put}^1(z)) = z \quad z : 0 \mid - \vdash \mathsf{put}^i(\mathsf{put}^j(z)) = \mathsf{put}^j(z) \tag{12}$$

$$x_0 : 0, x_1 : 0 \mid - \vdash \mathsf{put}^i(\mathsf{get}(x_0, x_1)) = \mathsf{put}^i(x_i) \tag{13}$$

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

$$x : 0 \mid - \vdash \mathsf{local}^i(a.\mathsf{close}(a, x)) = x \tag{14}$$

$$x_0 : 1, x_1 : 1 \mid - \vdash \mathsf{local}^i(a.\mathsf{get}(x_0(a),\ x_1(a))) = \mathsf{local}^i(a.x_i(a)) \tag{15}$$

$$z : 1 \mid - \vdash \mathsf{local}^i(a.\mathsf{put}^j(z(a))) = \mathsf{local}^j(a.z(a)) \tag{16}$$

$$z : 0 \mid a \vdash \mathsf{put}^i(\mathsf{close}(a, z)) = \mathsf{close}(a, z) \tag{17}$$

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

4 Models of parameterized theories

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

Models of first-order algebraic theories [2] consist 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 §2.2 we described the use of $\mathbf{Set}^{\mathbb{N}}$ in [23]. We connect the two approaches in Thm. 1 below.

At first glance, a term $x_1 : m_1, \ldots, x_k : m_k \mid a_1, \ldots, a_p \vdash t$ should denote a function $X(m_1) \times \ldots \times X(m_k) \to X(p)$, since a k-tuple of possible continuations that consume different numbers of parameters is mapped to a computation that consumes p parameters. However, the admissible substitution rule (8) shows us that actually such a term must also denote a sequence of functions

$$[\![x_1 : m_1, \ldots, x_k : m_k \mid a_1, \ldots, a_p \vdash t]\!]_{\mathcal{X},n} : X(n+m_1)\times\ldots\times X(n+m_k) \to X(n+p).$$

Definition 5. *Let Σ be a parameterized signature (Def. 3). 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*

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

The interpretation of terms is now defined by structural recursion in a standard way, where the interpretation of a computation variable term such as $x_1 : m_1, \ldots, x_k : m_k \mid a_1, \ldots, a_{m_i} \vdash x_i(a_1, \ldots, a_{m_i})$ is given by the sequence of product projections

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

Definition 6. *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 2. *The derivable equality ($=_{\mathcal{T}}$) in a parameterized algebraic theory $\mathcal{T}$ is sound: every $\mathcal{T}$-model satisfies every equation of $=_{\mathcal{T}}$.*

Proof (notes). By induction on the structure of derivations.

Remark 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 [31,14,35]. This is 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, §6], or make connections with syntactic categories [37]. Recall that the category $\mathbf{Set}^{\mathbb{N}}$ has a 'Day convolution' monoidal structure [6]: $(X \otimes Y)(n) = \sum_{m_1+m_2=n} X(m_1) \times Y(m_2)$. With this structure, we can interpret a parameterized algebraic theory $\mathcal{T}$ in any $\mathbf{Set}^{\mathbb{N}}$-enriched category $\mathcal{C}$ with products, powers, and copowers. A $\mathcal{T}$-model in $\mathcal{C}$ comprises an object $X \in \mathcal{C}$ 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 $\mathbf{y}(m) := \mathbb{N}(m, -)$, and $(A \cdot X)$ and $[A, X]$ for the copower and power.) The elementary notion of model (Def. 6) is recovered because, for the symmetric monoidal closed structure on $\mathbf{Set}^{\mathbb{N}}$ itself, $([\mathbf{y}(m), X])(n) = X(n+m)$. This also connects with (3), since $(\triangleright X) = \mathbf{y}(1) \otimes X$ and $(\triangleleft X) = [\mathbf{y}(1), X]$.

4.2 Free models and monads

Strong monads are of fundamental importance to computational effects [21]. Algebraic theories give rise to strong monads via free models.

In slightly more detail, 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.

Informally, for a theory $\mathcal{T}$ we define $F_{\mathcal{T}}X \in \mathbf{Set}^{\mathbb{N}}$ 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 $X(m_i)$. More formally, we let

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

where the equivalence relation $\sim$ allows us to α-rename context variables in the term judgements and apply permutation, contraction or weakening to the computation context paired with the corresponding transformation of the tuple $c_1, \dots, c_k$. It is straightforward to make $F_{\mathcal{T}}X$ into a Σ-structure.

Proposition 3.

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 Rem. 4.

Proof (notes). For (3), the monadic unit introduces variables and the bind is substitution. (In fact, this is part of an equivalence between such sifted-colimit-preserving strong monads and parameterized theories, e.g. [38, §5].)

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

In the case of a theory without equations, we recover exactly the scoped monad of Prop. 1 that was first given in [23]:

Theorem 1. *Consider signatures for algebraic Σ and scoped Σ' effects with no equations, inducing a parameterized algebraic theory $\mathcal{T}$ (via Rem. 2). We have an isomorphism of monads $F_{\mathcal{T}} \cong \left(\bar{\Sigma} + \left(\bar{\Sigma}' \circ \triangleleft\right) + \triangleright\right)^*$.*

Proof (notes). To see this, we use the description of $F_{\mathcal{T}}X(n)$ as a set of equivalence classes of $\mathcal{T}$-terms with computation variables coming from X. Consider the outermost operation of such a term: each of $\bar{\Sigma}$, $\left(\bar{\Sigma}' \circ \triangleleft\right)$ and $\triangleright$ on the right-hand-side corresponds to one of the three possibilities for this operation, algebraic, scoped or close respectively. Scoped operations bind a parameter and close consumes a parameter, hence the need for $\triangleleft/\triangleright$ on the right-hand-side: $\triangleleft$ increases the index n by 1 and $\triangleright$ decreases it, in keeping with Def. 5. Both $\triangleleft/\triangleright$ are characterized in Rem. 4 in terms of the Day tensor of $\mathbf{Set}^{\mathbb{N}}$.

4.3 Free models for scoped effects

We now turn to some concrete models from [23]. To characterize them as certain free models of parameterized algebraic theories, we need the following notion.

Definition 7. $X \in \mathbf{Set}^{\mathbb{N}}$ *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 [23], where if the programmer opens a scope, a matching closing of the scope is implicitly part of the program.

Nondeterminism. Recall the parameterized theory for nondeterminism with once (signature in Ex. 6 and equations in Fig. 2). It follows from Prop. 3 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:

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

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 ($[]$):

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

In fact the model $T_{\mathbf{o}}(X)$ we just described is the same as the model for nondeterminism from [23, Ex. 4.2]:

Theorem 2. *The model for nondeterminism with* once *from [23, Ex. 4.2], starting from a set A, is the free model on $\restriction A \in \mathbf{Set}^{\mathbb{N}}$ for the parameterized theory of nondeterminism with* once *(Fig. 2).*

Proof (notes). We obtain a description of the free model by directing the equations from Fig. 2 and computing the normal forms. Then we specialize to $\restriction A$.

Exceptions. Recall the parameterized theory of throwing and catching exceptions introduced in Ex. 7 and (9–11). 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 $T_{\mathbf{c}}(X)(n+1) \cong T_{\mathbf{c}}(X)(n) + \{e_{n+1}\}$. Below, x is an element of $T_{\mathbf{c}}(X)(n)$, standing for a computation in normal form:

$$\mathsf{catch}_n : T_{\mathbf{c}}(X)(n+1) \times T_{\mathbf{c}}(X)(n+1) \to T_{\mathbf{c}}(X)(n)$$
$$\mathsf{catch}_n(x, -) = x, \;\; \mathsf{catch}_n(e_{n+1}, x) = x, \;\; \mathsf{catch}_n(e_{n+1}, e_{n+1}) = e_n$$

$$\mathsf{close}_n : T_{\mathbf{c}}(X)(n) \to T_{\mathbf{c}}(X)(n+1) \qquad \mathsf{close}_n(x) = x$$
$$\mathsf{throw}_n : 1 \to T_{\mathbf{c}}(X)(n) \qquad \mathsf{throw}_n() = e_n$$

The cases in the definition of catch_n correspond to equations (11), (9), (10) respectively. In the third case, an exception inside $n+1$ scopes in the second argument of catch becomes an exception inside n scopes.

Theorem 3. *The model for exception catching from [23, Ex. 4.5], which starts from a set A, is the free model on $\restriction A \in \mathbf{Set}^{\mathbb{N}}$ for the parameterized theory of exceptions (9–11).*

State with local values. Recall the parameterized theory of mutable state with local values in Ex. 8 and its equations (12–17). The free model, in the sense of Prop. 3, on a truncated $X \in \mathbf{Set}^{\mathbb{N}}$ has carrier:

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

The operations on this model are

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

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 (12–16), omitting (17), the carrier of the free model on a truncated $X \in \mathbf{Set}^{\mathbb{N}}$ is:

$$T'_{\mathbf{l}}(X)(0) = \mathbb{2} \Rightarrow X(0) \times \mathbb{2} = T_{\mathbf{l}}(X)(0), \quad T'_{\mathbf{l}}(X)(n+1) = \mathbb{2} \Rightarrow T'_{\mathbf{l}}(X)(n) \times \mathbb{2}$$

In fact, $T'_{\mathbf{l}}(X)$ is the model of state with local proposed in [23, §7.1]:

Theorem 4. *Consider the example of state with local variables from [23], specialized to one memory location storing one bit, reading the return type a as a set A. The model proposed in [23, §7.1] is the free model on $\upharpoonright A$ for the parameterized algebraic theory with equations 12–16.*

The interpretations in $T_1(X)$ and $T_1'(X)$ (i.e that of [23]) of programs with no open scopes agree:

Proposition 4. *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:*

$$[\![t]\!]_{T_1(X),0} = [\![t]\!]_{T_1'(X),0} : T_1(X)(0)^n \to T_1(X)(0),$$

under the identification $T_1(X)(0) = T_1'(X)(0)$.

The restrictions of Γ to computation variables that do not depend on parameters and of Δ to be empty are reasonable because in the framework of [23], only programs with no open scopes are well-formed. Therefore, only such programs can be substituted in t, justifying the restriction of $[\![t]\!]_{T_1'(X)}$ to index 0.

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 (Rem. 2). As parameterized algebraic theories have a sound and complete algebraic theory (Props. 2, 3), this carries over to a sound and complete equational theory for scoped effects. We showed that our fresh perspective recovers the earlier models for scoped non-determinism, exceptions, and state (Thms. 2–4).

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. [25],[38, §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.

Our fresh perspective 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 [23]. For example, the parameterized algebraic theory of qubits [38] 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.

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

References

1. Abramsky, S., Jung, A.: Domain theory. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science, vol. 3 (1994)
2. Bauer, A.: What is algebraic about algebraic effects and handlers? (2019). https://doi.org/10.48550/arXiv.1807.05923
3. Benton, N., Wadler, P.: Linear logic, monads and the lambda calculus. In: Proceedings 11th Annual IEEE Symposium on Logic in Computer Science. pp. 420–431 (1996). https://doi.org/10.1109/LICS.1996.561458
4. Bizjak, A., Grathwohl, H.B., Clouston, R., Møgelberg, R.E., Birkedal, L.: Guarded dependent type theory with coinductive types. In: Jacobs, B., Löding, C. (eds.) Foundations of Software Science and Computation Structures. Lecture Notes in Computer Science, vol. 9634, p. 20–35. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49630-5_2
5. Bosman, R., van den Berg, B., Tang, W., Schrijvers, T.: A calculus for scoped effects & handlers (2023). https://doi.org/10.48550/arXiv.2304.09697
6. Day, B.: On closed categories of functors. In: MacLane, S., Applegate, H., Barr, M., Day, B., Dubuc, E., Phreilambud, Pultr, A., Street, R., Tierney, M., Swierczkowski, S. (eds.) Reports of the Midwest Category Seminar IV. pp. 1–38. Springer Berlin Heidelberg, Berlin, Heidelberg (1970)
7. Fiore, M.P., Staton, S.: Substitution, jumps, and algebraic effects. In: Proc. CSL-LICS2014 (2014)
8. Fiore, M., Szamozvancev, D.: Formal metatheory of second-order abstract syntax. Proc. ACM Program. Lang. **6**(POPL), 1–29 (2022). https://doi.org/10.1145/3498715, https://doi.org/10.1145/3498715
9. Fiore, M.P., Hur, C.: Second-order equational logic (extended abstract). In: Dawar, A., Veith, H. (eds.) Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6247, pp. 320–335. Springer (2010). https://doi.org/10.1007/978-3-642-15205-4_26, https://doi.org/10.1007/978-3-642-15205-4_26
10. Fiore, M.P., Mahmoud, O.: Second-order algebraic theories - (extended abstract). In: Hlinený, P., Kucera, A. (eds.) Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6281, pp. 368–380. Springer (2010). https://doi.org/10.1007/978-3-642-15155-2_33, https://doi.org/10.1007/978-3-642-15155-2_33
11. Girard, J.: Linear logic. Theor. Comput. Sci. **50**, 1–102 (1987)
12. Hyland, M., Plotkin, G., Power, J.: Combining effects: Sum and tensor. Theor. Comput. Sci. **357**(1), 70–99 (Jul 2006). https://doi.org/10.1016/j.tcs.2006.03.013
13. Katsumata, S.y., McDermott, D., Uustalu, T., Wu, N.: Flexible presentations of graded monads. Proc. ACM Program. Lang. **6**(ICFP) (aug 2022). https://doi.org/10.1145/3547654, https://doi.org/10.1145/3547654
14. Kelly, G., Power, A.: Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. Journal of Pure and Applied Algebra **89**(1), 163–179 (1993). https://doi.org/https://doi.org/10.1016/0022-4049(93)90092-8, https://www.sciencedirect.com/science/article/pii/0022404993900928
15. Lawvere, F.W.: Functorial semantics of algebraic theories. Proceedings of the National Academy of Sciences **50**(5), 869–872 (1963). https://doi.org/10.1073/pnas.50.5.869

16. Linton, F.E.J.: Some aspects of equational categories. In: Eilenberg, S., Harrison, D.K., MacLane, S., Röhrl, H. (eds.) Proceedings of the Conference on Categorical Algebra. pp. 84–94. Springer Berlin Heidelberg, Berlin, Heidelberg (1966). https://doi.org/10.1007/978-3-642-99902-4_3
17. Melliès, P.A.: Local states in string diagrams. In: Dowek, G. (ed.) Rewriting and Typed Lambda Calculi. pp. 334–348. Springer International Publishing, Cham (2014)
18. Melliès, P.A.: Segal condition meets computational effects. In: 2010 25th Annual IEEE Symposium on Logic in Computer Science. pp. 150–159 (2010). https://doi.org/10.1109/LICS.2010.46
19. Milner, R.: Communicating and Mobile Systems: The π-calculus. Cambridge University Press, United States (May 1999)
20. Moggi, E.: Computational lambda-calculus and monads. In: Proceedings. Fourth Annual Symposium on Logic in Computer Science. pp. 14–23 (1989). https://doi.org/10.1109/LICS.1989.39155
21. Moggi, E.: Notions of computation and monads. Information and Computation **93**(1), 55 – 92 (1991). https://doi.org/https://doi.org/10.1016/0890-5401(91)90052-4, selections from 1989 IEEE Symposium on Logic in Computer Science
22. Petersen, L., Harper, R., Crary, K., Pfenning, F.: A type theory for memory allocation and data layout. In: POPL 2003 (2003)
23. Piróg, M., Schrijvers, T., Wu, N., Jaskelioff, M.: Syntax and semantics for operations with scopes. In: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science. p. 809–818. LICS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3209108.3209166
24. Pitts, A.M.: Structural recursion with locally scoped names. Journal of Functional Programming **21**(3), 235–286 (2011). https://doi.org/10.1017/S0956796811000116
25. Plotkin, G., Power, J.: Algebraic operations and generic effects. Applied Categorical Structures **11**, 69–94 (2003)
26. Plotkin, G., Power, J.: Notions of computation determine monads. In: Nielsen, M., Engberg, U. (eds.) Foundations of Software Science and Computation Structures, 5th International Conference. pp. 342–356. FOSSACS 2002, Springer (2002), https://doi.org/10.1007/3-540-45931-6_24
27. Plotkin, G., Power, J.: Computational effects and operations: An overview. Electr. Notes Theor. Comput. Sci. **73**, 149–163 (10 2004). https://doi.org/10.1016/j.entcs.2004.08.008
28. Plotkin, G., Pretnar, M.: Handling algebraic effects. Logical Methods in Computer Science **9**(4) (Dec 2013). https://doi.org/10.2168/lmcs-9(4:23)2013
29. Plotkin, G.D., Power, J.: Adequacy for algebraic effects. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. Lecture Notes in Computer Science, vol. 2030, pp. 1–24. Springer (2001), https://doi.org/10.1007/3-540-45315-6_1
30. Polakow, J.: Ordered linear logic and applications. Ph.D. thesis, USA (2001)
31. Power, J.: Enriched Lawvere theories. Theory and Applications of Categories **6**(7), 83–93 (1999)
32. Power, J.: Semantics for local computational effects. Electronic Notes in Theoretical Computer Science **158**, 355–371 (May 2006). https://doi.org/10.1016/j.entcs.2006.04.018
33. Stark, I.: Free-algebra models for the pi -calculus. Theor. Comput. Sci. **390**(2-3), 248–270 (2008). https://doi.org/10.1016/j.tcs.2007.09.024, https://doi.org/10.1016/j.tcs.2007.09.024

34. Staton, S., Stein, D., Yang, H., Ackerman, N.L., Freer, C., Roy, D.: The Beta-Bernoulli process and algebraic effects. In: Proc. ICALP 2018 (2018)
35. Staton, S.: An algebraic presentation of predicate logic - (extended abstract). In: FOSSACS 2013 (2013)
36. Staton, S.: Instances of computational effects: An algebraic perspective. In: LICS 2013 (2013)
37. Staton, S.: Freyd categories are enriched lawvere theories. Electronic Notes in Theoretical Computer Science **303**, 197–206 (2014). https://doi.org/https://doi.org/10.1016/j.entcs.2014.02.010, https://www.sciencedirect.com/science/article/pii/S157106611400036X, proceedings of the Workshop on Algebra, Coalgebra and Topology (WACT 2013)
38. Staton, S.: Algebraic effects, linearity, and quantum programming languages. In: POPL 2015 (2015)
39. Thomson, P., Rix, R., Wu, N., Schrijvers, T.: Fusing industry and academia at GitHub (experience report). Proc. ACM Program. Lang. **6**(ICFP) (aug 2022). https://doi.org/10.1145/3547639, https://doi.org/10.1145/3547639
40. van den Berg, B., Schrijvers, T.: A framework for higher-order effects & handlers (2023). https://doi.org/10.48550/arXiv.2302.01415
41. Wu, N., Schrijvers, T., Hinze, R.: Effect Handlers in Scope. pp. 1–12 (2014). https://doi.org/10.1145/2633357.2633358
42. Yang, Z., Paviotti, M., Wu, N., van den Berg, B., Schrijvers, T.: Structured handling of scoped effects. p. 462–491. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-030-99336-8_17
43. Yang, Z., Wu, N.: Modular models of monoids with operations. Proc. ACM Program. Lang. **7**(ICFP) (aug 2023). https://doi.org/10.1145/3607850

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.