Parameterized algebraic theories and applications

Cristina Matache

University of Edinburgh

Introduction: algebraic effects

Side effects (e.g. I/O, state, nondeterminism) in functional languages can be modelled using:

- **monads** [Moggi'91] E.g. monads in Haskell;
- or using **algebraic theories** [Plotkin&Power]:
  - operations (e.g. reading and writing to memory) produce the effects;
  - program equations specify the behaviour of operations.

There is a correspondence between strong **monads** and **algebraic theories**.

Introduction: algebraic effects

Some effects do not fit into this algebraic framework.

Examples: dynamically allocating new memory locations, exception catching, continuations, normalizing a probability distribution, measuring qubits etc.

The following frameworks capture some of these effects:

- **Scoped effects** [Wu et. al.'14].
- **Parameterized algebraic theories** [Staton'13].

Contributions

- Scoped effects as parameterized theories
  [Lindley, Matache, Moss, Staton, Wu, Yang ESOP'24].
- Work in progress towards a parameterized theory of Unix fork.

Outline

1. Algebraic effects and algebraic theories
2. Parameterized algebraic theories
3. Scoped effects
4. Scoped effects as parameterized algebraic theories
5. A parameterized theory of threads (work in progress)

Example: explicit nondeterminism (backtracking) [Plotkin&Pretnar'09, '13]

Operations

$\mathsf{or}(x, y)$	choice
fail	failure

Equations

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

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

Generic effects:

$$\underline{\mathsf{or}} : \mathsf{unit} \to \mathsf{bool}$$

$$\underline{\mathsf{fail}} : \mathsf{unit} \to 0$$

$\mathsf{bool} = \text{arity} \qquad \mathsf{unit} = \text{coarity}$

Translation between generic effects and algebraic operations:

$$\mathsf{or}(x, y) = \mathsf{if}\ \underline{\mathsf{or}}()\ \mathsf{then}\ x\ \mathsf{else}\ y \qquad \underline{\mathsf{or}}() = \mathsf{or}(\mathsf{true}, \mathsf{false})$$

Example: explicit nondeterminism (backtracking) [Plotkin&Pretnar'09, '13]

Operations		Equations
$\mathsf{or}(x, y)$	choice	$x, y, z \vdash \mathsf{or}(x, \mathsf{or}(y, z)) = \mathsf{or}(\mathsf{or}(x, y), z)$
fail	failure	$x \vdash \mathsf{or}(\mathsf{fail}, x) = \mathsf{or}(x, \mathsf{fail}) = x$

Model = a set (carrier) + interpretations for the operations.

Fix a set A. The intended model for the theory of nondeterminism is:

- Carrier: the set $\mathsf{List}(A)$
- Operations:

$$\llbracket \mathsf{or} \rrbracket : \mathsf{List}(A)^2 \to \mathsf{List}(A), \qquad \llbracket \mathsf{or} \rrbracket(l_1, l_2) = l_1 @ l_2$$

$$\llbracket \mathsf{fail} \rrbracket : 1 \to \mathsf{List}(A), \qquad \llbracket \mathsf{fail} \rrbracket() = []$$

Example: one-bit state

Operations

$\mathsf{put}(i;\ x),\ i \in 2 = \{0,1\}$ writing

$\mathsf{get}(x_0, x_1)$ reading

Equations

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

$$x \vdash \mathsf{put}(i;\ \mathsf{put}(i';\ x)) = \mathsf{put}(i';\ x)$$

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

Generic effects:

$\underline{\mathsf{put}} : \mathsf{bool} \to \mathsf{unit}$ $\mathsf{unit} = \mathsf{arity}$ $\mathsf{bool} = \mathsf{coarity}$

$\underline{\mathsf{get}} : \mathsf{unit} \to \mathsf{bool}$

Example: one-bit state

Operations

$\mathsf{put}(i;\, x),\ i \in 2 = \{0, 1\}$ writing

$\mathsf{get}(x_0, x_1)$ reading

Equations

$$x_0, x_1 \vdash \mathsf{put}(i;\, \mathsf{get}(x_0, x_1)) = \mathsf{put}(i;\, x_i)$$
$$x \vdash \mathsf{put}(i;\, \mathsf{put}(i';\, x)) = \mathsf{put}(i';\, x)$$
$$x \vdash \mathsf{get}(\mathsf{put}(0;\, x), \mathsf{put}(1;\, x)) = x$$

Fix a set A. The intended **model** for the algebraic theory of state is:

- Carrier: $(A \times 2)^2$
- Operations:

$$[\![\mathsf{put}]\!] : 2 \times (A \times 2)^2 \to (A \times 2)^2 \qquad [\![\mathsf{put}]\!](i, f) = \lambda b.\, f(i)$$

$$[\![\mathsf{get}]\!] : (A \times 2)^2 \times (A \times 2)^2 \to (A \times 2)^2 \qquad [\![\mathsf{get}]\!](f_0, f_1) = \lambda b. \begin{cases} (f_0\, b) & \text{if } b = 0 \\ (f_1\, b) & \text{if } b = 1 \end{cases}$$

Algebraic theories

- Algebraic theories provide an **equational reasoning system** for algebraic effects, with semantic **models**, such that equality in the theory is **sound and complete**.
- $\mathsf{List}(A)$ and $(A \times 2)^2$ are **free models** on the set A.
- List and $(- \times 2)^2$ extend to strong monads on Set. Used for implementation and denotational semantics.
- Correspondence between **algebraic theories** and finitary (strong) **monads** on Set, via the free model construction.

Outline

1. Algebraic effects and algebraic theories
2. Parameterized algebraic theories
3. Scoped effects
4. Scoped effects as parameterized algebraic theories
5. A parameterized theory of threads (work in progress)

Parameterized theory example: local state

Local state = dynamically creating memory locations that store one bit

Parameters = location names

$\mathsf{put}(a, i;\ x)$	write value $i \in \{0, 1\}$ to location a, continue as x; a is a free parameter
$\mathsf{get}(a;\ x_0,\ x_1)$	read the bit stored in location a and continue as either x_0 or x_1; a is a free parameter
$\mathsf{new}(i;\ a.x(a))$	create a new location a, containing $i \in \{0, 1\}$ a is a fresh parameter, bound

$+$ operation for equality testing and equations [Staton LICS'13]

Parameterized theory example: local state

Parameters = location names

$\mathbb{P}$ = **abstract** type of parameters

Generic effects:

$$\mathsf{put}(a, i;\ x) \qquad \underline{\mathsf{put}} : \mathbb{P} \times \mathsf{bool} \to \mathsf{unit}$$

$$\mathsf{get}(a;\ x_0,\ x_1) \qquad \underline{\mathsf{get}} : \mathbb{P} \to \mathsf{bool}$$

$$\mathsf{new}(i;\ a.x(a)) \qquad \underline{\mathsf{new}} : \mathsf{bool} \to \mathbb{P} \qquad \mathbb{P} = \text{arity} \qquad \mathsf{bool} = \text{coarity}$$

For algebraic theories, arities and coarities are sums of unit. Now we allow sums and products with $\mathbb{P}$.

Parameterized algebraic theories [Staton FOSSACS'13, LICS'13, POPL'15]

- Uniform framework for axiomatizing local effects.
- Extend plain algebraic theories with **binding**.
- Provide an **equational reasoning system**, sound and complete with respect to **models**.
- Correspond to **monads** on a functor category.

Parameterized algebraic theories [Staton FOSSACS’13, LICS’13, POPL’15]

Example	Parameters	Models in
name generation	names	$\mathrm{Set}^{\mathrm{Fin}}$
local state	location names	
π-calculus (fragment)	communication channels	
first-order logic	individuals	
quantum computation	qubits (linear)	$\mathrm{Set}^{\mathrm{Bij}}$
scoped effects [ESOP’24]	scopes (ordered, linear)	$\mathrm{Set}^{\lvert\mathbb{N}\rvert}$
Unix fork (work in progess)	thread IDs	$\mathrm{Set}^{\mathrm{Fin}}$

For each of these functor categories: **parameterized theories** correspond to sifted-colimit-preserving strong **monads** (via the free model construction).

Outline

Example: explicit nondeterminism with once [Wu et. al.'14]

Operations:

$\mathsf{or}(x, y)$	choice
fail	failure
$\mathsf{once}(x)$	choose the first non-failing result of x

The **free model** uses the List monad. For a set A:

$$[\![\mathsf{or}]\!] : \mathsf{List}(A)^2 \to \mathsf{List}(A), \qquad [\![\mathsf{or}]\!](l_1, l_2) = l_1 @ l_2$$

$$[\![\mathsf{fail}]\!] : 1 \to \mathsf{List}(A), \qquad [\![\mathsf{fail}]\!]() = []$$

We want the interpretation of once to be:

$$[\![\mathsf{once}]\!] : \mathsf{List}(A) \to \mathsf{List}(A) \qquad [\![\mathsf{once}]\!]([a, \ldots]) = [a] \qquad [\![\mathsf{once}]\!]([]) = []$$

Algebraicity

Operations $[\![\mathsf{op}]\!]$ in the free model of a theory behave well with respect to the structure of the induced monad T:

$$[\![\mathsf{op}]\!] : TA \to TA \qquad\qquad \gg\!= : TA \times (A \Rightarrow TB) \to TB$$

$$\big([\![\mathsf{op}]\!](x) \gg\!= \lambda a.\, y\big) = [\![\mathsf{op}]\!](x \gg\!= \lambda a.\, y)$$

Example:

$$[\![\mathsf{or}]\!] : \mathsf{List}(\mathbb{N})^2 \to \mathsf{List}(\mathbb{N}) \qquad\qquad \gg\!= : \mathsf{List}(\mathbb{N}) \times (\mathbb{N} \Rightarrow \mathsf{List}(\mathbb{N})) \to \mathsf{List}(\mathbb{N})$$

$$k = \lambda n.\, [n, n+1] \qquad\qquad \big([\![\mathsf{or}]\!]([1],[3]) \gg\!= k\big) = \big([1,3] \gg\!= k\big) = [1,2,3,4]$$

$$[\![\mathsf{or}]\!]([1] \gg\!= k,\ [3] \gg\!= k) = [\![\mathsf{or}]\!]([1,2],[3,4]) = [1,2,3,4]$$

Algebraicity

Operations $[\![\mathsf{op}]\!]$ in the free model of a theory behave well with respect to the structure of the induced monad T:

$$[\![\mathsf{op}]\!] : TA \to TA \qquad\qquad \gg\!= : TA \times (A \Rightarrow TB) \to TB$$

$$\big([\![\mathsf{op}]\!](x) \gg\!= \lambda a.\, y\big) = [\![\mathsf{op}]\!](x \gg\!= \lambda a.\, y)$$

Example using the generic effect:

$$\underline{\mathsf{or}} : \mathsf{unit} \to \mathsf{bool}$$

$$\mathsf{if}\ \underline{\mathsf{or}}()\ \mathsf{then}\ f(); h()\ \mathsf{else}\ g(); h() = \big(\mathsf{if}\ \underline{\mathsf{or}}()\ \mathsf{then}\ f()\ \mathsf{else}\ g()\big); h()$$

Algebraicity fails for once

$[\![\text{once}]\!] : \mathsf{List}(A) \to \mathsf{List}(A)$ is **not algebraic** with respect to the List monad.

Example:

$$[\![\text{or}]\!] : \mathsf{List}(\mathbb{N})^2 \to \mathsf{List}(\mathbb{N}) \qquad \gg\!= : \mathsf{List}(\mathbb{N}) \times (\mathbb{N} \Rightarrow \mathsf{List}(\mathbb{N})) \to \mathsf{List}(\mathbb{N})$$

$$k = \lambda n.\, [n, n+1] \qquad \big([\![\text{once}]\!]([\![\text{or}]\!]([1],[3]))\big) \gg\!= k = \big([1] \gg\!= k\big) = [1,2]$$

$$[\![\text{once}]\!]\big([\![\text{or}]\!]([1],[3]) \gg\!= k\big) = [\![\text{once}]\!]([1,2,3,4]) = [1]$$

- once doesn’t present a monad in the usual sense
- Intuition: the scope of once is delimited, even though its arity is that of an algebraic operation. once is called a **scoped effect**.

Scoped effects: background

- Other examples: exception catching, state with local variables.
- Scoped effects implemented as **effect handlers** [Plotkin&Pretnar'09, '13]. Not guaranteed to satisfy equations.
- Treating scoped effects as operations: work on free monads from signatures and extending effect handlers to handle scoped operations. [Wu et. al.'14], [Piróg et.al. LICS'18], [Yang et.al. ESOP'22, ICFP'23]

Our contribution [Lindley, Matache, Moss, Staton, Wu, Yang ESOP'24]

Finding a notion of algebraic theory to axiomatize scoped effects.

Outline

Scoped effects as parameterized algebraic theories

Parameters = names of scopes

A scoped effect = a theory with **ordered, linear** parameters

Example: explicit nondeterminism with once

$\mathsf{or}(x, y)$	choice
fail	failure
$\mathsf{once}(a.x(a))$	open a new scope named a, continue as $x(a)$; a is bound
$\mathsf{close}(a;\ y)$	close the scope a, y cannot use a anymore; a is free

Closing a scope becomes an explicit operation. We write:

$\mathsf{once}\big(a.\mathsf{or}(\mathsf{close}(a;\ 1), \mathsf{close}(a;\ 3))\big)$ instead of $\mathsf{once}(\mathsf{or}(1, 3))$

Equations for explicit nondeterminism with once

Explicit nondeterminism

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

$$\mathsf{or}(\mathsf{fail}, x) = \mathsf{or}(x, \mathsf{fail}) = x$$

Once/close

$$\mathsf{once}(a.\mathsf{close}(a;\ x)) = x$$

$$\mathsf{once}(a.\mathsf{fail}) = \mathsf{fail}$$

$$\mathsf{once}(a.\mathsf{or}(x(a),\ x(a))) = \mathsf{once}(a.x(a))$$

$$\mathsf{once}\big(a.\mathsf{or}(\mathsf{close}(a;\ x),\ y(a))\big) = x$$

We can prove using the equations:

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

Models for explicit nondeterminism with once

Model = an object from $\mathsf{Set}^{|\mathbb{N}|}$ (carrier) + interpretations for the operations.

The **free model** on $A = (A_0, \emptyset, \emptyset, \dots) \in \mathsf{Set}^{|\mathbb{N}|}$ has:

- Carrier: the sequence $TA(n) = \mathsf{List}^{n+1}(A_0)$, for $n \in \mathbb{N}$
- Operations:

$$
\begin{array}{lll}
[\![\mathsf{once}]\!]_n : TA(n+1) \to TA(n) & [\![\mathsf{once}]\!]_n([a, \dots]) = a, & [\![\mathsf{once}]\!]_n([]) = [] \\
[\![\mathsf{close}]\!]_n : TA(n) \to TA(n+1) & [\![\mathsf{close}]\!]_n(l) = [l] & \\
[\![\mathsf{or}]\!]_n : TA(n)^2 \to TA(n) & [\![\mathsf{or}]\!]_n(l_1, l_2) = l_1 @ l_2 & \\
[\![\mathsf{fail}]\!]_n : 1 \to TA(n) & [\![\mathsf{fail}]\!]_n() = [] &
\end{array}
$$

Recall: $\mathsf{once}(a.x(a))$ $\quad$ $\mathsf{close}(a;\ y)$ $\quad$ $\mathsf{or}(x, y)$ $\quad$ fail

Scoped effects as parameterized algebraic theories

Theorem [ESOP'24]

The **free model** on $A = (A_0, \emptyset, \emptyset, \dots) \in \mathsf{Set}^{|\mathbb{N}|}$ (from previous slide) is the monad algebra used to model nondeterminism with once in [Piróg et.al. LICS'18].

- Therefore, we have an equational characterization of a model from the scoped effects literature.
- We have analogous results for exception catching and state with local variables.
- Parameterized theories allow sound and complete equational reasoning for scoped effects.

Outline

Applications of parameterized theories

Example	Parameters	Models in
name generation	names	$\mathsf{Set}^{\mathsf{Fin}}$
local state	location names	
π-calculus (fragment)	communication channels	
first-order logic	individuals	
quantum computation	qubits (linear)	$\mathsf{Set}^{\mathsf{Bij}}$
scoped effects [ESOP'24]	scopes (ordered,linear)	$\mathsf{Set}^{\vert\mathbb{N}\vert}$
Unix fork (work in progess)	thread IDs	$\mathsf{Set}^{\mathsf{Fin}}$

I'll discuss the example of forking threads. Parameters are unconstrained.

Parameterized theory of threads with names

Parameters = thread IDs = $\mathbb{P}$

Operations:

$\mathsf{fork}(a.x(a),\ y)$	x = parent thread; might use a y = child thread; cannot use a a = ID of child; bound name
$\mathsf{wait}(a;\ x)$	wait for the thread named a to finish, continue as x
stop	this thread has finished (no continuation)

Generic effects: $\underline{\mathsf{fork}} : \mathsf{unit} \to \mathbb{P} + \mathsf{unit}$ $\underline{\mathsf{wait}} : \mathbb{P} \to \mathsf{unit}$

(Tentative) Equations for the parameterized theory of threads

Children that stop immediately:

$$x : 0 \mid - \vdash \mathsf{fork}(a.x,\ \mathsf{stop}) = x \qquad x : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{wait}(a;\ x),\ \mathsf{stop}) = x$$

Forking and waiting commute:

$$x : 2, y : 1 \mid b \vdash \mathsf{fork}(a.\mathsf{wait}(b;\ x(b, a)),\ \mathsf{wait}(b;\ y(b))) = \mathsf{wait}(b;\ \mathsf{fork}(a.x(b, a),\ y(b)))$$

Waiting is idempotent and commutative:

$$x : 1 \mid a \vdash \mathsf{wait}(a;\ \mathsf{wait}(a;\ x(a))) = \mathsf{wait}(a;\ x(a))$$

$$x : 2 \mid a, b \vdash \mathsf{wait}(a;\ \mathsf{wait}(b;\ x(a, b))) = \mathsf{wait}(b;\ \mathsf{wait}(a;\ x(a, b)))$$

(Tentative) Equations for the parameterized theory of threads

Forking is commutative and associative:

$$x : 2, y_1, y_2 : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{fork}(b.x(a, b),\ y_2),\ y_1) = \mathsf{fork}(b.\mathsf{fork}(a.x(a, b),\ y_1),\ y_2)$$

$$x, y : 1, z : 0 \mid - \vdash \mathsf{fork}(a.x(a),\ \mathsf{fork}(b.y(b),\ z)) = \mathsf{fork}(b.\mathsf{fork}(a.x(a),\ y(b)),\ z)$$

The parent might stop before its children:

$$y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{stop},\ y) \neq y$$

But:

$$y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{wait}(a;\ \mathsf{stop}), y) = y$$

Parameterized theory of threads – Work in progress

- Find a nice description of the free model for this theory (without quotienting by equations), and hence of a monad.
- Compare the equations with an operational semantics for threads. Change the equations if needed.

Goal

A sound and adequate denotational semantics for Unix threads, using a monad.

Summary and future work

Summary:

- Parameterized algebraic theories extend algebraic theories with more arities (formed from an abstract type $\mathbb{P}$).
- They can be used to reason equationally about:
  - scoped effects (operations that are not “algebraic”)
  - forking threads (work in progress)

Future work:

- Effect handlers for parameterized operations
- For scoped effects
  - Axiomatize more examples e.g. backtracking with cut
  - Programming with generic effects