Parameterized algebraic theories and computational effects

Cristina Matache

University of Edinburgh

Introduction

Denotational semantics of programming languages is about finding models of programs in order to:

- reason about programs,
- and to compare them more easily.
- We want models to be compositional.

We often use:

- the simply-typed λ-calculus as a core programming language;
- category theory as a tool for organizing models.

Introduction

These methods work well for functional programming: a program is a function that takes an input and returns a value.

But programs interact with the environment e.g.:

- manipulating memory,
- taking input from a keyboard,
- probabilistic computation.

These are known as **computational effects**.

We want to **reason** about them **compositionally**. Monads and algebraic theories help us give a unified semantics to effects.

Outline

A setting for denotational semantics of effects [Moggi, LICS'89]

- A Cartesian category $\mathcal{C}$ (distinguished terminal object and binary products),
- a strong monad T on $\mathcal{C}$,
- and Kleisli exponentials:
  for each $B, C \in \mathsf{Ob}(\mathcal{C})$ an isomorphism

$$\mathcal{C}(A \times B,\, TC) \cong \mathcal{C}(A,\, B \Rightarrow TC) \qquad \text{natural in } A$$

  for some specified object $B \Rightarrow TC$.

This structure is enough to model a variant of simply-typed lambda calculus suitable for studying effects.

Examples of strong monads (on Set)

Each monad models a different effect:

- One bit of memory: $TX = 2 \Rightarrow (X \times 2)$;
- Exceptions, from a set E: $TX = X + E$;
- Binary nondeterminism: finite powerset monad.

We might use monads on other categories depending on what other features the PL has e.g. the category of $\omega\mathsf{cpo}$'s.

Sketching a calculus for studying effects

The setting: $\mathcal{C}$ cartesian category, T a strong monad, Kleisli exponentials, can model **fine-grain call-by-value** λ-calculus [Levy, Power, Thielecke'03]:

- There are two kinds of programs, those that don't perform effects (pure), and those that do (effectful):

$$\Gamma \vdash^{\mathsf{pure}} V : A \qquad\qquad \Gamma \vdash^{\mathsf{eff}} M : A$$

- Types A are interpreted as objects in $\mathcal{C}$.
- Γ is a list of types, interpreted using the cartesian structure of $\mathcal{C}$.
- Functions of type $A \to B$ are interpreted using Kleisli exponentials $[\![A]\!] \Rightarrow T[\![B]\!]$

Sketching a calculus for studying effects

The setting: $\mathcal{C}$ cartesian category, T a strong monad, Kleisli exponentials, can model **fine-grain call-by-value** λ-calculus [Levy, Power, Thielecke'03]:

- There are two kinds of terms, those that don't perform effects (pure), and those that may (effectful):

$$\Gamma \vdash^{\mathsf{pure}} V : A \qquad\qquad \Gamma \vdash^{\mathsf{eff}} M : A$$

- Terms are interpreted as morphisms

$$[\![\Gamma \vdash^{\mathsf{pure}} V : A]\!] : [\![\Gamma]\!] \to [\![A]\!] \qquad \text{in } \mathcal{C}$$

$$[\![\Gamma \vdash^{\mathsf{eff}} M : A]\!] : [\![\Gamma]\!] \to [\![A]\!] \qquad \text{in } \mathsf{Kl}(T) \text{ (i.e. } [\![\Gamma]\!] \to T[\![A]\!] \text{ in } \mathcal{C}\text{)}$$

- One can show that the equational laws of the calculus, e.g. β/η, are satisfied by the interpretation (soundness).

For each effect, we adapt the calculus

Recall the monad for one bit of memory: $TX = 2 \Rightarrow (X \times 2)$.

To write programs that manipulate the state, we add the following term constructors:

$$\frac{\Gamma \vdash^{\mathsf{eff}} M_0 : A \qquad \Gamma \vdash^{\mathsf{eff}} M_1 : A}{\Gamma \vdash^{\mathsf{eff}} \mathsf{get}(M_0, M_1) : A} \qquad \frac{\Gamma \vdash^{\mathsf{eff}} M : A}{\Gamma \vdash^{\mathsf{eff}} \mathsf{put}_0(M) : A} \qquad \frac{\Gamma \vdash^{\mathsf{eff}} M : A}{\Gamma \vdash^{\mathsf{eff}} \mathsf{put}_1(M) : A}$$

- get reads the bit in memory, if it's 0 continues as M_0, otherwise as M_1.
- put_0 writes 0 to memory, continues as M.

(The syntax makes $\mathcal{C}(\llbracket - \rrbracket, TA)$ a get/put algebra, so by Yoneda TA (sort of) is an internal get/put algebra.)

Sequencing effectful terms is crucial [Levy, Power, Thielecke'03]

Using explicit sequencing in the calculus

$$\frac{\Gamma \vdash^{\mathsf{eff}} M : A \qquad \Gamma, x : A \vdash^{\mathsf{eff}} N : B}{\Gamma \vdash^{\mathsf{eff}} \mathsf{let}\, x = M \,\mathsf{in}\, N : B}$$

we specify exactly the evaluation order we want. Evaluation order matters:

$$M_1 \stackrel{\text{def}}{=} \mathsf{let}\, x = \mathsf{put}_1(\mathsf{return}\, 42) \,\mathsf{in}\, \mathsf{let}\, y = \mathsf{get}(\mathsf{return}\, 0, \mathsf{return}\, 1) \,\mathsf{in}\, \mathsf{return}\, (x, y)$$

$$M_2 \stackrel{\text{def}}{=} \mathsf{let}\, y = \mathsf{get}(\mathsf{return}\, 0, \mathsf{return}\, 1) \,\mathsf{in}\, \mathsf{let}\, x = \mathsf{put}_1(\mathsf{return}\, 42) \,\mathsf{in}\, \mathsf{return}\, (x, y)$$

M_1 always returns $(42, 1)$. M_2 returns what is already in memory, $(42, ?)$.
get and put don't commute with each other.

Sequencing effectful terms is crucial [Levy, Power, Thielecke'03]

Explicit sequencing in the calculus

$$\frac{\Gamma \vdash^{\mathsf{eff}} M : A \qquad \Gamma, x : A \vdash^{\mathsf{eff}} N : B}{\Gamma \vdash^{\mathsf{eff}} \mathsf{let}\, x = M \,\mathsf{in}\, N : B}$$

is interpreted using the **strength** of the monad T

$$\begin{aligned} &[\![\mathsf{let}\, x = M \,\mathsf{in}\, N]\!] \\ &\quad : [\![\Gamma]\!] \xrightarrow{\Delta} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{\mathsf{id} \times [\![M]\!]} [\![\Gamma]\!] \times T[\![A]\!] \xrightarrow{\mathsf{str}} T([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{T[\![N]\!]} TT[\![B]\!] \\ &\qquad \xrightarrow{\mu} T[\![B]\!] \end{aligned}$$

Outline

Presentations of algebraic theories are fundamental in PL

"Computational effects determine monads but are not identified with monads. We regard a computational effect as being realised by families of operations, with a monad being generated by their equational theory."

[Plotkin & Power, *Notions of computation determine monads*, '02]

An algebraic theory determines a monad by the free-algebra construction and:

Theorem [Lawvere, Linton]

To give a finitary monad on the category of sets is equivalent to giving an algebraic theory.

(Where "algebraic theory" is a suitable presentation-independent notion.)

Example: a presentation for one bit of memory

The monad on Set that models one bit of state

$$TX = 2 \Rightarrow (X \times 2)$$

is generated by the following operations and equations, where $i, i' \in \{1, 0\}$,

$$\mathsf{put}_i(\mathsf{get}(x_0, x_1)) = \mathsf{put}_i(x_i)$$

$$\mathsf{put}_i(\mathsf{put}_{i'}(x)) = \mathsf{put}_{i'}(x)$$

$$\mathsf{get}(\mathsf{put}_0(x), \mathsf{put}_1(x)) = x$$

which are computationally natural.

(The equational theory can be extended to account for a fixed set of memory locations, and for storing an infinite datatype e.g. $\mathbb{N}$ [Plotkin & Power'02].)

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

Question

Can we extend the syntactic framework of algebraic theories to axiomatize other effects?
And still derive a denotational semantics for a fine-grain λ-calculus with effects via monads.

Example: dynamically allocating new memory locations that store one bit.

Dynamically allocated memory

Let $\mathbb{L}$ be a base type of memory locations.

$$\frac{(a : \mathbb{L}) \in \Gamma \qquad \Gamma \vdash^{\mathsf{eff}} M_0 : A \qquad \Gamma \vdash^{\mathsf{eff}} M_1 : A}{\Gamma \vdash^{\mathsf{eff}} \mathsf{get}(a; M_0, M_1) : A}$$

$$\frac{(a : \mathbb{L}) \in \Gamma \qquad \Gamma \vdash^{\mathsf{eff}} M : A}{\Gamma \vdash^{\mathsf{eff}} \mathsf{put}_i(a; M) : A}(i \in \{0, 1\}) \qquad \frac{\Gamma, a : \mathbb{L} \vdash^{\mathsf{eff}} M : A}{\Gamma \vdash^{\mathsf{eff}} \nu_i a.\, M : A}(i \in \{0, 1\})$$

$$\frac{(a, b : \mathbb{L}) \in \Gamma \qquad \Gamma \vdash^{\mathsf{eff}} M_0 : A \qquad \Gamma \vdash^{\mathsf{eff}} M_1 : A}{\Gamma \vdash^{\mathsf{eff}} M_0 ?_{a=b} M_1 : A}$$

$\nu_i a.\, M$ creates a new memory location a, storing i, that is bound in M.

$?_{a=b}$ checks if a and b are the same location, if yes run M_0, o.w. run M_1.

Dynamically allocated memory

The effect operations can be axiomatized using a **presentation** of a **parameterized algebraic theory**. This has two kinds of variables:

- a, stands for memory locations, can be bound;
- x stands for continuations.

Example equations (18 equations in total, see [Staton, LICS'13]):

$$a \vdash \nu_i b.(x(b) \, ?_{a=b} \, y(b)) = \nu_i b.\, y(b)$$

$$a, b \vdash \mathsf{put}_i(a; \mathsf{get}(b; x_0, x_1)) = \mathsf{put}_i(a; x_i) \, ?_{a=b} \, \mathsf{get}(b; \mathsf{put}_i(a; x_0), \mathsf{put}_i(a; x_1))$$

The equations generate a strong monad on $\mathsf{Set}^{\mathsf{Fin}}$.
Fin = skeleton of the category of finite sets and all functions.

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

- The theory of dynamically allocated state has **models** in $\mathsf{Set}^{\mathsf{Fin}}$:
  - where the interpretation of the location type is fixed:

$$[\![\mathbb{L}]\!] = \mathsf{Fin}(1, -) = y(1).$$

  - An object n of Fin is a world with n memory locations.
- Provide a syntax for equationally axiomatizing "local" effects.
- Provide an equational reasoning system that is **sound and complete** w.r.t. models.

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

A **model** is a functor $X \in \mathsf{Ob}(\mathsf{Set}^{\mathsf{Fin}})$, together with an interpretation for the operations e.g.:

$$[\![\mathsf{get}]\!] : X^2 \to X^{[\![\mathbb{L}]\!]} \qquad [\![\mathsf{put}_i]\!] : X \to X^{[\![\mathbb{L}]\!]} \qquad [\![\nu_i]\!] : X^{[\![\mathbb{L}]\!]} \to X$$

$$[\![?_=]\!] : X^2 \to X^{[\![\mathbb{L}]\!] \times [\![\mathbb{L}]\!]}$$

such that the equations are satisfied, where $[\![\mathbb{L}]\!] = \mathsf{Fin}(1, -)$.

Compare this with the type of $[\![\mathsf{get}]\!] : X^2 \to X$ for a single one-bit memory location. Instead of finite sums of 1, the arities and coarities can be sums and products of $\mathsf{Fin}(1, -)$.

Variations of parameterized theories [Staton]

Varying the indexing category, and the syntax of the equational theory:

Example	Parameters (typed as $y(1)$)	Models in
name generation	names	$\mathsf{Set}^{\mathsf{Fin}}$
local memory	location names	
π-calculus (fragment)	communication channels	
first-order logic	individuals	
probabilistic programming	urns	
substitution	code pointers	
quantum computation	qubits (linear)	$\mathsf{Set}^{\mathsf{Bij}}$
scoped effects [TOPLAS'25]	scopes (ordered, linear)	$\mathsf{Set}^{\lvert\mathbb{N}\rvert}$
idealized POSIX threads [POPL'26]	thread IDs	$\mathsf{Set}^{\mathbb{S}^{op}}$

($\mathbb{S}$ is a suitably chosen Lawvere theory.)

Parameterized algebraic theories

Example	Parameters (typed as $y(1)$)	Models in
name generation	names	$\mathsf{Set}^{\mathsf{Fin}}$
local memory	location names	
π-calculus (fragment)	communication channels	
first-order logic	individuals	
probabilistic programming	urns	
substitution	code pointers	
quantum computation	qubits (linear)	$\mathsf{Set}^{\mathsf{Bij}}$
scoped effects [TOPLAS'25]	scopes (ordered, linear)	$\mathsf{Set}^{\lvert\mathbb{N}\rvert}$
idealized POSIX threads [POPL'26]	thread IDs	$\mathsf{Set}^{\mathbb{S}^{op}}$

Theorem [Staton'13; Lack & Rosicky'11, *Notions of Lawvere theories*]

For each case in the table:

to give a parameterized algebraic theory is to give a sifted-colimit preserving strong monad on the respective functor category.

(In the Bij and $\lvert\mathbb{N}\rvert$ cases the strength is w.r.t. the Day convolution monoidal structure.)

Outline

Forking threads and waiting for them [Kammar, Liell-Cock, Lindley, Matache, Staton]

tid is a base type of thread IDs, only introduced by fork.

$$\frac{\Gamma, a : \mathsf{tid} \vdash^{\mathsf{eff}} M_1 : A \qquad \Gamma \vdash^{\mathsf{eff}} M_2 : A}{\Gamma \vdash^{\mathsf{eff}} \mathsf{fork}(a.M_1,\ M_2) : A} \qquad \frac{(a : \mathsf{tid}) \in \Gamma \qquad \Gamma \vdash^{\mathsf{eff}} M : A}{\Gamma \vdash^{\mathsf{eff}} \mathsf{wait}(a; M) : A}$$

$$\frac{}{\Gamma \vdash^{\mathsf{eff}} \mathsf{stop} : A}$$

$\mathsf{fork}(a.M_1,\ M_2)$ creates a new thread M_2 that runs **concurrently** with M_1. The parent, M_1, has the ID, a, of the child M_2.

$\mathsf{wait}(a; M)$ blocks until the thread a to finishes, then does M.

stop ends the thread, unblocks all other threads waiting for it.

A parameterized theory of threads

We axiomatized fork, wait and stop using 8 equations, for example:

- wait and fork commute

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

- $\mathsf{wait}(a; \mathsf{stop})$ acts as a unit for fork

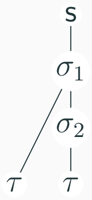
$$\vdash \mathsf{fork}(a.\mathsf{wait}(a; \mathsf{stop}), x) = x$$

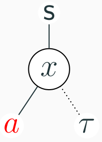
$$a \vdash \mathsf{fork}(b.x(b), \mathsf{wait}(a; \mathsf{stop})) = x(a)$$

giving a parameterized theory with models in $\mathsf{Set}^{\mathsf{FinRel}}$.
FinRel = finite sets and relations.

A parameterized theory of threads

The theory of fork/wait/stop induces a monad on $\mathsf{Set}^{\mathsf{FinRel}}$.
We give a **representation theorem** for the monad in terms of partially-ordered multisets.





We use the monad to give a denotational semantics to a fine-grain λ-calculus with threads.

We show the denotational semantics matches an operational semantics.

Summary

- A setting for denotational semantics of λ-calculus with effects: $\mathcal{C}$ cartesian, T strong monad, Kleisli exponentials.
- Strength is crucial for sequencing effects.
- Presentations of algebraic theories capture the essence of effects, e.g. memory; they determine monads.
- Parameterized theories generalize plain algebraic theories:
  - they axiomatize more sophisticated effects e.g. dynamically allocated memory, threads;
  - they have an equational reasoning system,
  - and have correspondence to monads on functor categories.