

Algebraizing Higher-Order Effects

Martin Andrieux Alan Schmitt

Université de Rennes INRIA

Running Example: A Simple State Language

Syntax

- Expressions: $e ::= n \mid x \mid e + e \mid \text{read}(n)$
- Commands: $c ::= \text{write}(n, e) \mid \text{let } x = e \text{ in } c \mid c_1; c_2 \mid \text{skip}$

Example

```
write(0, 12);  
let x = read(0) in  
write(1, x + 1)
```

Example

```
write(0, 12);  
(let x = read(0) in skip);  
write(1, x + 1)
```

Monadic Evaluator (Haskell)

```
evalE :: Expression -> M Int
evalE (Lit n)      = return n
evalE (Var x)      = ask x
evalE (Plus e1 e2) = do {v1 <- evalE e1; v2 <- evalE e2; return (v1 + v2)}
evalE (Read a)     = get a
```

```
evalC :: Command -> M ()
evalC (Write a e) = do {v <- evalE e; set a v}
evalC (Let x e c) = do {v <- evalE e; local (x,v) (evalC c)}
evalC (Seq c1 c2) = do {evalC c1; evalC c2}
```

Type of Computations

The monad `M` provides the *get*, *set*, *ask*, and *local* effects without committing to a particular implementation.

Why Do we Need Effects?

Impure Features in Pure Languages

- Pure languages are unable to deal with mutable memory, exceptions, ...
- Clean separation between logic and implementation

Different usages, various interpretations

- Labelled Transition Systems
- Reasoning with an equational theory
- Efficient implementation (get/set = load/store)

The Algebraic Ideal

- **Algebraic Effects** (Plotkin & Power) are abstract operations $op(e, p \mapsto s)$.

The Algebraic Property

An operation op is algebraic if it composes with any continuation k :

$$op(e, p \mapsto s) \gg= k = op(e, p \mapsto s \gg= k)$$

- **Why it is great:**
 - Algebraic effects can be interpreted via *handlers*
 - Programs can be seen as *trees of operations* (free monads, interaction trees)
 - Program transformations (like CPS) are *simpler*

Does `local` satisfy this?

Scoped Effects Are not Algebraic

- Scoped effects (`local`, `catch`) take a *sub-computation* as an argument
- They do **not** compose with sequencing

The Case of `local`

```
triple = do
  r <- ask           -- r = 1
  x <- local (2 * r) ( --
    ask             --
  )                 -- x = 2
  y <- ask           -- y = 1
  return (x + y)    --

triple? = do
  r <- ask           -- r = 1
  local (2 * r) ( do --
    x <- ask         -- x = 2
    y <- ask         -- y = 2
    return (x + y)  --
  )
```

$\text{op}(\text{local}_r, p \mapsto \text{ret } ()) \gg \text{ask} \neq \text{op}(\text{local}_r, p \mapsto \text{ret } ()) \gg \text{ask}$

Existing Approaches to Scoped Effects

- Scoped effect as **effect handlers**:
 - In the algebraic effects framework: higher-order effects = handlers
 - This sacrifices the separation between syntax and semantics
- **Scoped Effects** (Wu et al., 2014):
 - Higher-order free monads, tree of trees
 - \implies Standard tools (like `itrees`) need to be heavily modified
 - Open/Close approach put aside

Potential Benefit of Open/Close

- $((\text{local } r) \ c1 \ \gg \ c2)$ becomes $(\text{open } (\text{local } r) \ \gg \ c1 \ \gg \ \text{close} \ \gg \ c2)$
- Algebraic effects only, easy manipulation
- But loss of structural information: where is the scoped computation?

Adapting Open/Close

- **Scoped Effects as Parameterized Algebraic Theories** (Matache et al., 2025):
 - Focus on well-bracketing of Open/Close operations (to keep structural information)
 - The interpretation of an effects depends on its depth in the scope nesting
- **Our Work** (The paper):

HO Effects

```
local :: r -> m b -> (b -> m a) -> m a
```

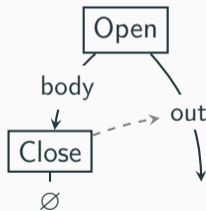
Algebraic Effects

```
local :: r -> m a -> (b -> m a) -> m a
```

```
open  :: r -> ((() + b) -> m a) -> m a
```

```
close :: b -> m Void
```

⇒ More structural info but harder to call the continuation



Semantics Preservation

- Given a set of HO effects Ho and an interpretation Interp_{Ho}
- For each HO computation C , we consider the Open/Close computation $\text{Alg}(C)$
- Are we able to interpret $\text{Alg}(C)$ “the right way”?

Contribution

$$\forall \text{Interp}_{Ho} \exists \text{Interp}_{Alg} \forall C, \text{Interp}_{Ho}(C) = \text{Interp}_{Alg}(\text{Alg}(C))$$

- **Proven in Agda** for (get/set + ask/local + throw/catch) effects (≈ 400 lines)
- **Idea:** rebuild the original term C from $\text{Alg}(C)$ (i.e., $\text{Interp}_{Ho} = id$)

Interpreting Algebraized Effects Directly

What Should Interpretation Do?

- HO effects (and scoped effects) are about altering control flow (e.g., `catch`)
- Interpretation orchestrates continuations (run *body*, if *exn* run *handler*, run *out*)
- We must be able to capture the current continuation and resume it later

Problems that Remains to Be Solved

1. We cannot interpret `itrees` in the continuation monad
2. We use free monads, but the reasoning tools available to `itree` are missing

Conclusion & Future work

- **Target:** The Rocq/Coq backend for Skel
- **Strategy:** Use “Algebraization” to bypass the complexity of HO effects

The Workflow

1. **Compile** Skel HO effects to `Open/Close` algebraic effects
2. **Embed** into `itrees` (Interaction Trees) or free monads in Rocq
3. **Prove** that the shallow embedding matches the intended HO semantics

Long term goals:

- Support a large set of effects, or support *any* effect
- Automatic generation of the shallow embedding (`rocq_of_skel`)