Algebraic Effects and Handlers

Ningning Xie

University of Toronto
OPLSS 2025

Summary

An algebraic effect is given by

- operations (i.e. *effect constructors*) with signatures
- a set of axioms

Effect handlers

- give interpretations to operations
- effects and handlers can be nested and composed

2. Effect handlers

2.2. Effect type system

Effect type system

expressions

$e := c \mid x \mid \lambda x.\ e \mid e_1\ e_2 \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ e \mid \textbf{handle}\ h\ e$

$v := c \mid \lambda x.\ e$

$h := \{\mathit{op} \mapsto \lambda x\ k.\ e_1, \mathit{return} \mapsto \lambda x.\ e_2\}$

Effect type system

expressions

$e := c \mid x \mid \lambda x.\ e \mid e_1\ e_2 \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ e \mid \textbf{handle}\ h\ e$

$v := c \mid \lambda x.\ e$

$h := \{\mathit{op} \mapsto \lambda x\ k.\ e_1, \mathit{return} \mapsto \lambda x.\ e_2\}$

effect labels ℓ example: state has get and put

Effect type system

expressions

$e := c \mid x \mid \lambda x.\ e \mid e_1\ e_2 \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ e \mid \textbf{handle}\ h\ e$

$v := c \mid \lambda x.\ e$

$h := \{\mathit{op} \mapsto \lambda x\ k.\ e_1, \mathit{return} \mapsto \lambda x.\ e_2\}$

effect labels ℓ example: state has get and put

effects $\epsilon = \langle\rangle \mid \langle \ell \mid \epsilon \rangle$

effects as: ➤ sets (Eff) ➤ simple rows (Links) ➤ scoped rows (Koka)

Effect type system

expressions

$e := c \mid x \mid \lambda x.\ e \mid e_1\ e_2 \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ e \mid \textbf{handle}\ h\ e$

$v := c \mid \lambda x.\ e$

$h := \{\mathit{op} \mapsto \lambda x\ k.\ e_1, \mathit{return} \mapsto \lambda x.\ e_2\}$

effect labels ℓ example: state has get and put

effects $\epsilon = \langle\rangle \mid \langle \ell \mid \epsilon \rangle$

effects as: ➤ sets (Eff) ➤ simple rows (Links) ➤ scoped rows (Koka)

types $A := \mathit{Int} \mid \mathit{Bool} \mid A \to \epsilon\ B$

- Read $A \to \epsilon\ B$ as: a function that, when given A, may perform effects in ϵ, and returns B

Effect type system

$$\boxed{\Gamma \vdash e : A \mid \epsilon}$$

$$\frac{}{\Gamma \vdash \textbf{True} : \mathit{Bool} \mid \epsilon'}\ \mathrm{T} \qquad \frac{}{\Gamma \vdash \textbf{False} : \mathit{Bool} \mid \epsilon'}\ \mathrm{F} \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A \mid \epsilon'}\ \mathrm{VAR}$$

$$\frac{\Gamma, x : A \vdash e : B \mid \epsilon}{\Gamma \vdash \lambda x.\ e : A \to_{\epsilon} B \mid \epsilon'}\ \mathrm{LAM} \qquad \frac{\Gamma \vdash e_1 : A \to_{\epsilon} B \mid \epsilon \qquad \Gamma \vdash e_2 : A \mid \epsilon}{\Gamma \vdash e_1\ e_2 : B \mid \epsilon}\ \mathrm{APP}$$

$$\frac{\Gamma \vdash e_1 : \mathit{Bool} \mid \epsilon \qquad \Gamma \vdash e_2 : A \mid \epsilon \qquad \Gamma \vdash e_3 : A \mid \epsilon}{\Gamma \vdash \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 : A \mid \epsilon}\ \mathrm{IF}$$

Effect type system

$$\boxed{\Gamma \vdash e : A \mid \epsilon}$$

$$\frac{\mathit{op} : A \rightsquigarrow B \in \Sigma(\ell) \qquad \Gamma \vdash e : A \mid \langle \ell \mid \epsilon \rangle}{\Gamma \vdash \mathit{op}\ e : B \mid \langle \ell \mid \epsilon \rangle}\ \textsc{op}$$

$$\frac{\Gamma \vdash e : C \mid \langle \ell \mid \epsilon \rangle \quad \Gamma \vdash h : C \Rightarrow \langle \ell \mid \epsilon \rangle D}{\Gamma \vdash \textbf{handle}\ h\ e : D \mid \epsilon}\ \textsc{handle}$$

Effect type system

$$\boxed{\Gamma \vdash e : A \mid \epsilon}$$

$$\frac{\mathit{op} : A \rightsquigarrow B \in \Sigma(\ell) \qquad \Gamma \vdash e : A \mid \langle \ell \mid \epsilon \rangle}{\Gamma \vdash \mathit{op}\ e : B \mid \langle \ell \mid \epsilon \rangle}\ \text{OP}$$

$$\frac{\Gamma \vdash e : C \mid \langle \ell \mid \epsilon \rangle \quad \Gamma \vdash h : C \Rightarrow \langle \ell \mid \epsilon \rangle D}{\Gamma \vdash \textbf{handle}\ h\ e : D \mid \epsilon}\ \text{HANDLE}$$

$$\boxed{\Gamma \vdash h : C \Rightarrow \langle \ell \mid \epsilon \rangle D}$$

$$\frac{\Gamma, x : C \vdash e : D \mid \epsilon \qquad \mathit{op}_i : A_i \rightsquigarrow B_i \in \Sigma(\ell) \qquad \Gamma, x_i : A_i, k_i : B_i \to \epsilon D \vdash e_i : D \mid \epsilon}{\Gamma \vdash \{\mathit{return} \mapsto \lambda x.\ e,\ \mathit{op}_i \mapsto \lambda x_i\ k_i.\ e_i\} : C \Rightarrow \langle \ell \mid \epsilon \rangle D}\ \text{HND}$$

Type safety

Theorem (Type Preservation)
If $\Gamma \vdash e : A \mid \epsilon$, *and* $e \longrightarrow e'$, *then* $\Gamma \vdash e' : A \mid \epsilon$.

Type safety

Theorem (Type Preservation)
If $\Gamma \vdash e : A \mid \epsilon$, *and* $e \longrightarrow e'$, *then* $\Gamma \vdash e' : A \mid \epsilon$.

Theorem (Progress)
If $\bullet \vdash e : A \mid \epsilon$, *then either* e *is a value, or there exists* e' *such that* $e \longrightarrow e'$.

Type safety

Theorem (Type Preservation)
If $\Gamma \vdash e : A \mid \epsilon$, *and* $e \longrightarrow e'$, *then* $\Gamma \vdash e' : A \mid \epsilon$.

Theorem (Progress)
If $\bullet \vdash e : A \mid \epsilon$, *then either* e *is a value, or there exists* e' *such that* $e \longrightarrow e'$.

Question: what if e performs an operation in ϵ?

Type safety

Theorem (Type Preservation)
If $\Gamma \vdash e : A \mid \epsilon$, *and* $e \longrightarrow e'$, *then* $\Gamma \vdash e' : A \mid \epsilon$.

Theorem (Progress with effects)
If $\bullet \vdash e : A \mid \epsilon$, *then either* e *is a value, or there exists* e' *such that* $e \longrightarrow e'$, *or* $e = E[op\ v]$, *where* $op \in \epsilon$ *and* $op \# E$.

Type safety

Theorem (Type Preservation)
If $\Gamma \vdash e : A \mid \epsilon$, *and* $e \longrightarrow e'$, *then* $\Gamma \vdash e' : A \mid \epsilon$.

Theorem (Progress with effects)
If $\bullet \vdash e : A \mid \epsilon$, *then either* e *is a value, or there exists* e' *such that* $e \longrightarrow e'$, *or* $e = E[op\ v]$, *where* $op \in \epsilon$ *and* $op \# E$.

Corollary (Progress)
If $\bullet \vdash e : A \mid \langle\rangle$, *then either* e *is a value, or there exists* e' *such that* $e \longrightarrow e'$.

Algebraic effects vs monads

Haskell'19

Monad Transformers and Modular Algebraic Effects

What Binds Them Together

Tom Schrijvers
KU Leuven
Belgium

Maciej Piróg
University of Wrocław
Poland

Nicolas Wu
Imperial College London
United Kingdom

Mauro Jaskelioff
CIFASIS-CONICET
Universidad Nacional de Rosario
Argentina

Abstract

For over two decades, monad transformers have been the main *modular* approach for expressing purely functional side-effects in Haskell. Yet, in recent years algebraic effects have emerged as an alternative whose popularity is growing.

While the two approaches have been well-studied, there is still confusion about their relative merits and expressiveness, especially when it comes to their comparative modularity. This paper clarifies the connection between the two approaches—some of which is folklore—and spells out consequences that we believe should be better known.

We characterise a class of algebraic effects that is modular, and show how these correspond to a specific class of monad transformers. In particular, we show that our modular algebraic effects gives rise to monad transformers. Moreover, every monad transformer for algebraic operations gives rise to a modular effect handler.

CCS Concepts • **Software and its engineering → Functional languages**; *Control structures*; *Coroutines*; • **Theory of computation** → *Categorical semantics.*

Keywords Handlers, Effects, Monads, Transformers

ACM Reference Format:
Tom Schrijvers, Maciej Piróg, Nicolas Wu, and Mauro Jaskelioff. 2019. Monad Transformers and Modular Algebraic Effects: What

1 Introduction

For decades monads [29, 42] have dominated the scene of pure functional programming with effects, and the recent popularisation of algebraic effects & handlers [3, 7, 21, 23, 35] promises to change the landscape. However, with rapid change also comes confusion, and practitioners have been left uncertain about the advantages and limitations of the two competing approaches. This paper aims at clarifying the essential differences and similarities between them.

Working with combinations of multiple different effects demands a *modular* approach, where each effect is given semantics separately. This allows for the construction of complex custom effects from off-the-shelf building blocks.

A popular approach to achieving modularity for monads is with monad transformers [27]. Monad transformers extend an arbitrary monad with a new effect while at the same time ensuring that original effects are available. The desired combination of monads is achieved by stacking several monad transformers in a particular order.

In the algebraic effects approach modularity is conceptually achieved in two stages. First, the syntax of all operations involved in the effect are defined. Then a program is incrementally interpreted by several handlers, which in turn give the syntax of different effects a semantics.

Since each handler only knows about the part of the syntax of the effect it is handling, a modular approach to algebraic

3. Implementing effect handlers

Expensive runtime operations

$$\textbf{handle}\ h\ E[op\ v] \longrightarrow f\ v\ (\lambda x.\ \textbf{handle}\ h\ E[x]) \quad \text{where } op \mapsto f \in h, op \# E$$

Expensive runtime operations

Two potentially expensive runtime operations:

$$\textbf{handle}\ h\ E[op\ v] \longrightarrow f\ v\ \overbrace{(\lambda x.\ \textbf{handle}\ h\ E[x])}^{\textit{capturing}} \quad \text{where } op \mapsto f \in h,\ \overbrace{op \# E}^{\textit{searching}}$$

Expensive runtime operations

Two potentially expensive runtime operations:

$$\textbf{handle}\ h\ E[op\ v] \longrightarrow f\ v\ \overbrace{(\lambda x.\ \textbf{handle}\ h\ E[x])}^{\textit{capturing}} \quad \text{where } op \mapsto f \in h,\ \overbrace{op \# E}^{\textit{searching}}$$

$$\begin{aligned} &\mathit{allChoices} = \{ \\ &\quad \mathit{choose} \mapsto \lambda x\ k.\ k\ \textbf{True} +\!\!+ k\ \textbf{False}, \\ &\quad \mathit{return} \mapsto \lambda x.\ [x]\ \} \end{aligned}$$

$$\textbf{handle}\ \mathit{allChoices}\ 42 \longrightarrow [42]$$
$$\textbf{handle}\ \mathit{allChoices}\ (\mathit{toss}\ ()) \longrightarrow [\mathit{Heads}, \mathit{Tails}]$$

Compile effect handlers efficiently

- Continuation-passing style

  Closure allocation cost

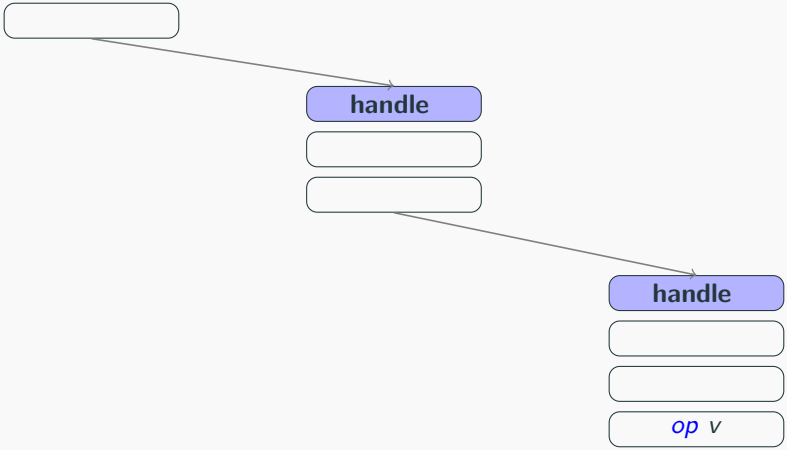
Links [Hillerström et al.(2017)]

Compile effect handlers efficiently

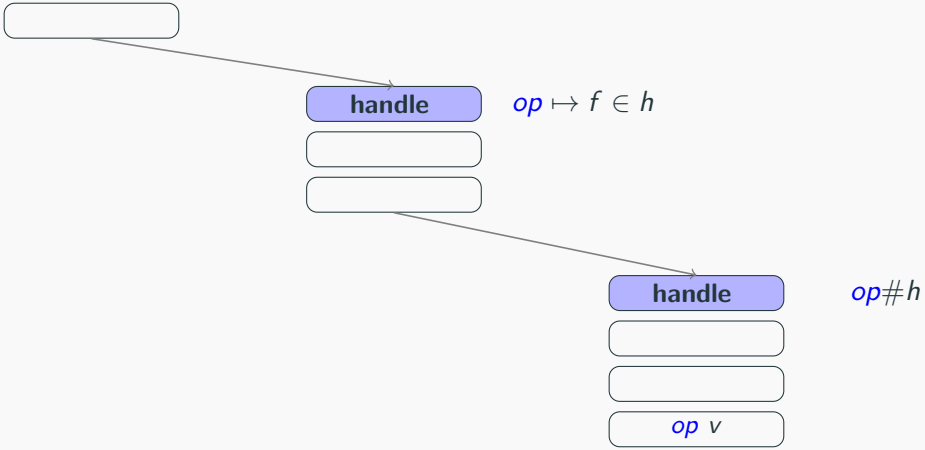
- Continuation-passing style — Links [Hillerström et al.(2017)]
  - Closure allocation cost
- Segmented stacks — OCaml [Sivaramakrishnan et al.(2021)]
  - Efficient one-shot resumption

Segmented stacks

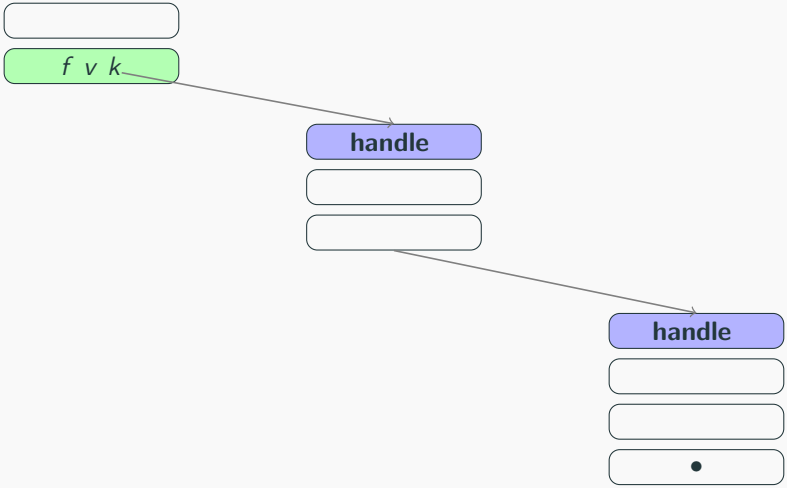
Segmented stacks



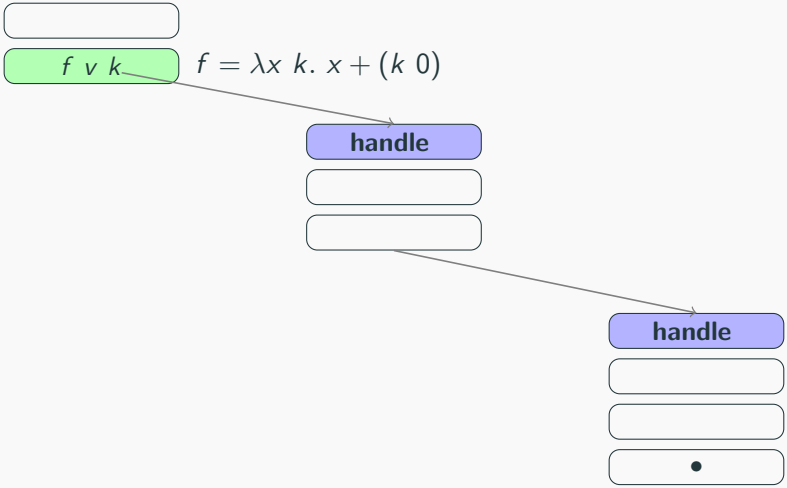
Segmented stacks



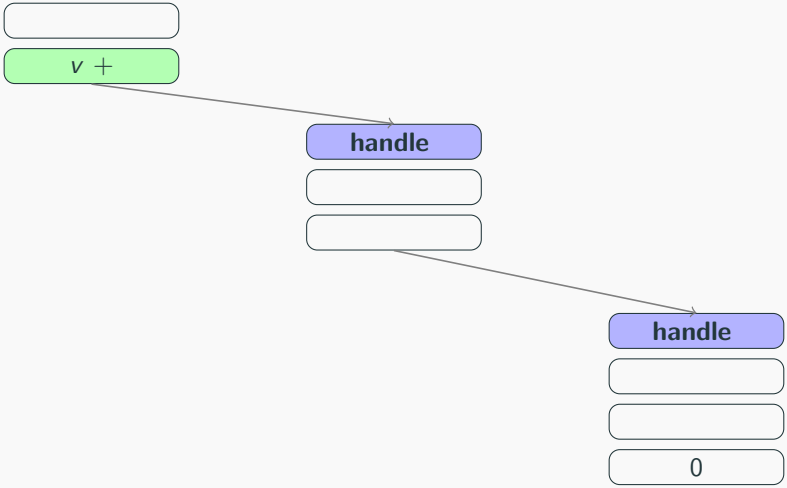
Segmented stacks



Segmented stacks



Segmented stacks



Compile effect handlers efficiently

- Continuation-passing style — Links [Hillerström et al.(2017)]
  - Closure allocation cost
- Segmented stacks — OCaml [Sivaramakrishnan et al.(2021)]
  - Efficient one-shot resumption

Compile effect handlers efficiently

- Continuation-passing style — Links [Hillerström et al.(2017)]
  - Closure allocation cost
- Segmented stacks — OCaml [Sivaramakrishnan et al.(2021)]
  - Efficient one-shot resumption
- Capability-passing style — Effekt [Schuster et al.(2020), Brachthäuser et al.(2020)]
  - Efficient lexically scoped handlers

Compile effect handlers efficiently

- Continuation-passing style — Links [Hillerström et al.(2017)]
  - Closure allocation cost
- Segmented stacks — OCaml [Sivaramakrishnan et al.(2021)]
  - Efficient one-shot resumption
- Capability-passing style — Effekt [Schuster et al.(2020), Brachthäuser et al.(2020)]
  - Efficient lexically scoped handlers
- Rewriting — Eff [Karachalias et al.(2021)]
  - Source-to-source transformations

Compile effect handlers efficiently

- Continuation-passing style — Links [Hillerström et al.(2017)]
  - Closure allocation cost
- Segmented stacks — OCaml [Sivaramakrishnan et al.(2021)]
  - Efficient one-shot resumption
- Capability-passing style — Effekt [Schuster et al.(2020), Brachthäuser et al.(2020)]
  - Efficient lexically scoped handlers
- Rewriting — Eff [Karachalias et al.(2021)]
  - Source-to-source transformations
- Evidence-passing semantics — Koka [Xie et al.(2020), Xie and Leijen(2021)]
  - Efficient tail-resumptive operations

Evidence-passing semantics

- Pass handlers down to the operation call sites
- Evaluate tail-resumptive operations in-place

ICFP'20

Effect Handlers, Evidently

NINGNING XIE, Microsoft Research, USA
JONATHAN IMMANUEL BRACHTHÄUSER, University of Tübingen, Germany
DANIEL HILLERSTRÖM, The University of Edinburgh, United Kingdom
PHILIPP SCHUSTER, University of Tübingen, Germany
DAAN LEIJEN, Microsoft Research, USA

Algebraic effect handlers are a powerful way to incorporate effects in a programming language. Sometimes perhaps even *too* powerful. In this article we define a restriction of general effect handlers with *scoped resumptions*. We argue one can still express all important effects, while improving reasoning about effect handlers. Using the newly gained guarantees, we define a sound and coherent evidence translation for effect handlers, which directly passes the handlers as evidence to each operation. We prove full soundness and coherence of the translation into plain lambda calculus. The evidence in turn enables efficient implementations of effect operations; in particular, we show we can execute tail-resumptive operations *in place* (without needing to capture the evaluation context), and how we can replace the runtime search for a handler by indexing with a constant offset.

CCS Concepts: • **Software and its engineering → Control structures**; **Polymorphism**; • **Theory of computation → Type theory**.

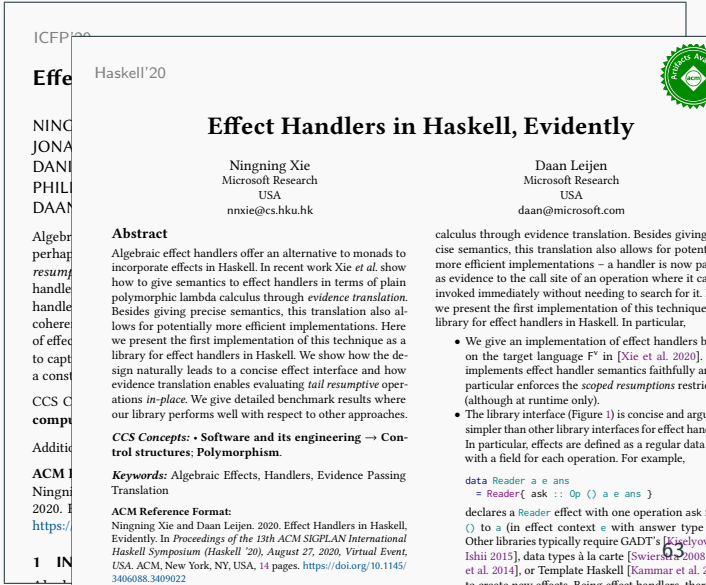
Additional Key Words and Phrases: Algebraic Effects, Handlers, Evidence Passing Translation

ACM Reference Format:
Ningning Xie, Jonathan Immanuel Brachthäuser, Daniel Hillerström, Philipp Schuster, and Daan Leijen. 2020. Effect Handlers, Evidently. *Proc. ACM Program. Lang.* 4, ICFP, Article 99 (August 2020), 29 pages. https://doi.org/10.1145/3408981

1 INTRODUCTION

Evidence-passing semantics

- Pass handlers down to the operation call sites
- Evaluate tail-resumptive operations in-place
- A Haskell library



ICFP'20

Haskell'20

Effect Handlers in Haskell, Evidently

Ningning Xie
Microsoft Research
USA
nnxie@cs.hku.hk

Daan Leijen
Microsoft Research
USA
daan@microsoft.com

Abstract

Algebraic effect handlers offer an alternative to monads to incorporate effects in Haskell. In recent work Xie *et al.* show how to give semantics to effect handlers in terms of plain polymorphic lambda calculus through *evidence translation*. Besides giving precise semantics, this translation also allows for potentially more efficient implementations. Here we present the first implementation of this technique as a library for effect handlers in Haskell. We show how the design naturally leads to a concise effect interface and how evidence translation enables evaluating *tail resumptive* operations *in-place*. We give detailed benchmark results where our library performs well with respect to other approaches.

CCS Concepts: • **Software and its engineering** → **Control structures**; **Polymorphism**.

Keywords: Algebraic Effects, Handlers, Evidence Passing Translation

ACM Reference Format:
Ningning Xie and Daan Leijen. 2020. Effect Handlers in Haskell, Evidently. In *Proceedings of the 13th ACM SIGPLAN International Haskell Symposium (Haskell '20), August 27, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3406088.3409022

calculus through evidence translation. Besides giving
cise semantics, this translation also allows for potent
more efficient implementations – a handler is now pa
as evidence to the call site of an operation where it ca
invoked immediately without needing to search for it.
we present the first implementation of this technique
library for effect handlers in Haskell. In particular,

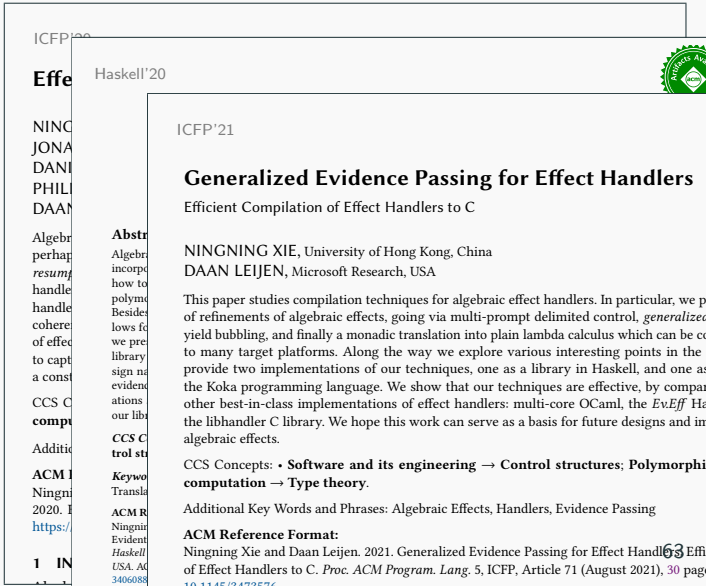
- We give an implementation of effect handlers b
on the target language F^c in [Xie et al. 2020].
implements effect handler semantics faithfully ar
particular enforces the *scoped resumptions* restri
(although at runtime only).
- The library interface (Figure 1) is concise and argu
simpler than other library interfaces for effect han
In particular, effects are defined as a regular data
with a field for each operation. For example,

```
data Reader a e ans
  = Reader{ ask :: Op () a e ans }
```

declares a Reader effect with one operation ask
() to a (in effect context e with answer type
Other libraries typically require GADT's [Kiselyov
Ishii 2015], data types à la carte [Swierstra 2008
et al. 2014], or Template Haskell [Kammar et al. 2

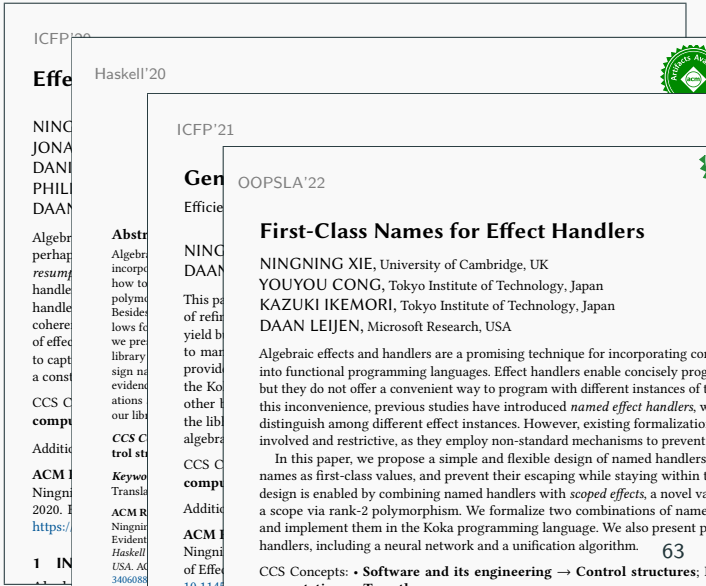
Evidence-passing semantics

- Pass handlers down to the operation call sites
- Evaluate tail-resumptive operations in-place
- ~~A Haskell library~~
  Two Haskell libraries



Evidence-passing semantics

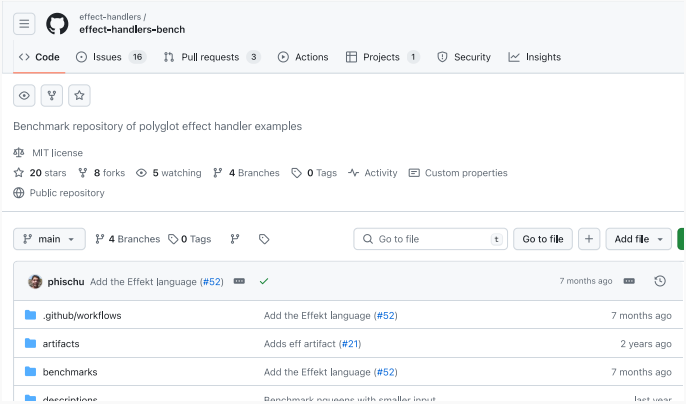
- Pass handlers down to the operation call sites
- Evaluate tail-resumptive operations in-place
- ~~A Haskell library~~
  Two Haskell libraries
- Specify the handler when performing



Compile effect handlers efficiently

- Continuation-passing style — Links [Hillerström et al.(2017)]
  - Closure allocation cost
- Segmented stacks — OCaml [Sivaramakrishnan et al.(2021)]
  - Efficient one-shot resumption
- Capability-passing style — Effekt [Schuster et al.(2020), Brachthäuser et al.(2020)]
  - Efficient lexically scoped handlers
- Rewriting — Eff [Karachalias et al.(2021)]
  - Source-to-source transformations
- Evidence-passing semantics — Koka [Xie et al.(2020), Xie and Leijen(2021)]
  - Efficient tail-resumptive operations

Effect handlers benchmarks suite



https://github.com/effect-handlers/effect-handlers-bench

Resources

- Effect bibliography
  `https://github.com/yallop/effects-bibliography`
- "An introduction to algebraic effects and handlers" [Pretnar(2015)]
- "Effect-Handler Oriented Programming" [Lindley, OPLSS'22]
- "Programming with Effect Handlers and FBIP in Koka" [Leijen and Xie, ICFP'21 Tutorial]
- "What is algebraic about algebraic effects and handlers?" [Bauer(2018)]


Andrej Bauer. 2018.
What is algebraic about algebraic effects and handlers?
arXiv preprint arXiv:1807.05923 (2018).


Jonathan Immanuel Brachthäuser, Philipp Schuster, and Klaus Ostermann. 2020.
Effects as capabilities: effect handlers and lightweight effect polymorphism.
Proceedings of the ACM on Programming Languages 4, OOPSLA (2020), 1–30.


Daniel Hillerström, Sam Lindley, Robert Atkey, and K. C. Sivaramakrishnan. 2017.
Continuation Passing Style for Effect Handlers. In 2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017), Dale Miller (Ed.), Vol. 84. Dagstuhl, Germany, 18:1–18:19.
doi:10.4230/LIPIcs.FSCD.2017.18


Georgios Karachalias, Filip Koprivec, Matija Pretnar, and Tom Schrijvers. 2021.
Efficient compilation of algebraic effect handlers.
Proceedings of the ACM on Programming Languages 5, OOPSLA (2021), 1–28.


Matija Pretnar. 2015.
An introduction to algebraic effects and handlers. invited tutorial paper.
Electronic notes in theoretical computer science 319 (2015), 19–35.


Philipp Schuster, Jonathan Immanuel Brachthäuser, and Klaus Ostermann. 2020.
Compiling effect handlers in capability-passing style.
Proceedings of the ACM on Programming Languages 4, ICFP (2020), 1–28.


KC Sivaramakrishnan, Stephen Dolan, Leo White, Tom Kelly, Sadiq Jaffer, and Anil Madhavapeddy. 2021.
Retrofitting effect handlers onto OCaml. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. 206–221.


Ningning Xie, Jonathan Immanuel Brachthäuser, Daniel Hillerström, Philipp Schuster, and Daan Leijen. 2020.

Effect handlers, evidently.
Proceedings of the ACM on Programming Languages 4, ICFP (2020), 1–29.

Ningning Xie and Daan Leijen. 2021.
Generalized evidence passing for effect handlers: efficient compilation of effect handlers to C.
Proceedings of the ACM on Programming Languages 5, ICFP (2021), 1–30.