Effect handler oriented programming

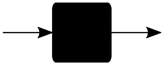
Sam Lindley

The University of Edinburgh

Huawei Strategy & Technology Workshop, September 2022

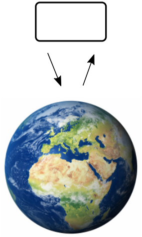
Effects

Programs as black boxes (Church-Turing model)?



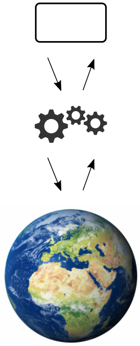
Effects

Programs must interact with their context



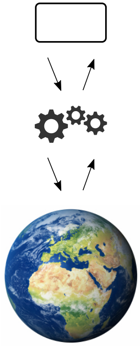
Effects

Programs must interact with their context



Effects

Programs must interact with their context

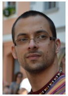


Effects are pervasive

- input/output
  user interaction
- concurrency
  web applications
- distribution
  cloud computing
- exceptions
  fault tolerance
- choice
  backtracking search

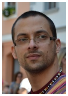
Typically ad hoc and hard-wired

Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009

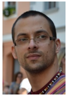
Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009

Composable and **customisable** user-defined interpretation of effects in general

Effect handlers

 Gordon Plotkin  Matija Pretnar

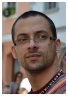
Handlers of algebraic effects, ESOP 2009

Composable and **customisable** user-defined interpretation of effects in general

Give programmer direct access to **context**

(c.f. resumable exceptions, monads, delimited control)

Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009

Composable and **customisable** user-defined interpretation of effects in general

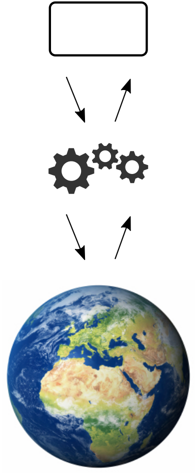
Give programmer direct access to **context**

(c.f. resumable exceptions, monads, delimited control)

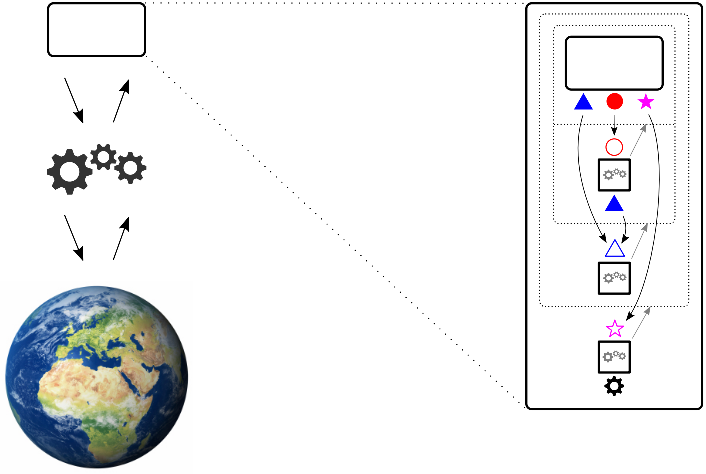
Growing industrial interest

GitHub	`semantic`	Code analysis library (> 25 million repositories)
	React	JavaScript UI library (> 2 million websites)
Uber	Pyro	Statistical inference (10% ad spend saving)

Effect handlers as composable user-defined operating systems



Effect handlers as composable user-defined operating systems



Example 1: generators

Effect signature

$$\{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}$$

Example 1: generators

Effect signature

$$\{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}$$

A simple generator

$$\begin{aligned}&\mathsf{nats} : \mathsf{Nat} \to 1!(e \,\&\, \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\})\\&\mathsf{nats}\; n = \mathsf{send}\; n; \mathsf{nats}\,(n+1)\end{aligned}$$

Example 1: generators

Effect signature

$$\{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}$$

A simple generator

$$\begin{aligned}
&\mathsf{nats} : \mathsf{Nat} \to 1!(e \,\&\, \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}) \\
&\mathsf{nats}\; n = \mathsf{send}\; n; \mathsf{nats}\,(n+1)
\end{aligned}$$

Handler

$$\begin{aligned}
&\mathsf{sumUntil} : \mathsf{Nat} \to (1 \to 1!(e \,\&\, \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\})) \to \mathsf{Nat}!e \\
&\mathsf{sumUntil}\; \mathit{stop}\; t = \\
&\quad \textbf{handle}\; t\,()\; \textbf{with} \\
&\qquad \textbf{return}\,() \qquad \mapsto 0 \\
&\qquad \langle \mathsf{send}\; n \to r \rangle \;\mapsto \textbf{if}\; n \le \mathit{stop}\; \textbf{then}\; n + \mathsf{sumUntil}\; \mathit{stop}\; r \\
&\qquad \qquad \qquad \qquad \;\; \textbf{else}\; 0
\end{aligned}$$

Example 1: generators

Effect signature

$$\{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}$$

A simple generator

$$\begin{aligned}
&\mathsf{nats} : \mathsf{Nat} \to 1!(e \,\&\, \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}) \\
&\mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\,(n+1)
\end{aligned}$$

Handler

$$\begin{aligned}
&\mathsf{sumUntil} : \mathsf{Nat} \to (1 \to 1!(e \,\&\, \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\})) \to \mathsf{Nat}!e \\
&\mathsf{sumUntil}\, \mathit{stop}\; t = \\
&\quad \mathbf{handle}\; t\,()\; \mathbf{with} \\
&\qquad \mathbf{return}\,() \quad \mapsto 0 \\
&\qquad \langle \mathsf{send}\; n \to r \rangle \mapsto \mathbf{if}\; n \le \mathit{stop}\; \mathbf{then}\; n + \mathsf{sumUntil}\, \mathit{stop}\; r \\
&\qquad\qquad\qquad\qquad\quad \mathbf{else}\; 0
\end{aligned}$$

$$\mathsf{sumUntil}\, 5\, (\lambda().\mathsf{nats}\, 0) \Longrightarrow 15$$

Example 2: lightweight threads

Effect signature

$$\{\text{yield} : 1 \twoheadrightarrow 1\}$$

Example 2: lightweight threads

Effect signature

$$\{\mathsf{yield} : 1 \twoheadrightarrow 1\}$$

Two cooperative lightweight threads

$$\begin{aligned}
\mathsf{tA}\,() &= \mathsf{print}\,(\text{“A1 ”});\, \mathsf{yield}\,();\, \mathsf{print}\,(\text{“A2 ”}) \\
\mathsf{tB}\,() &= \mathsf{print}\,(\text{“B1 ”});\, \mathsf{yield}\,();\, \mathsf{print}\,(\text{“B2 ”})
\end{aligned}$$

Example 2: lightweight threads

Types

$$\text{Thread } e = 1 \to 1!(e \,\&\, \{\text{yield} : 1 \twoheadrightarrow 1\})$$

Handler

$$\begin{aligned}
&\text{cooperate} : \text{List } (\text{Thread } e) \to 1!e \\
&\text{cooperate } [] = () \\
&\text{cooperate } (r :: rs) = \\
&\quad \textbf{handle } r() \textbf{ with} \\
&\quad\quad \textbf{return } () \qquad\quad \mapsto \text{cooperate } (rs) \\
&\quad\quad \langle \text{yield } () \to s \rangle \mapsto \text{cooperate } (rs +\!\!+ [s])
\end{aligned}$$

Example 2: lightweight threads

Types

$$\text{Thread } e = 1 \to 1!(e \,\&\, \{\text{yield} : 1 \twoheadrightarrow 1\})$$

Handler

$$\begin{aligned}
&\text{cooperate} : \text{List } (\text{Thread } e) \to 1!e \\
&\text{cooperate } [] = () \\
&\text{cooperate } (r :: rs) = \\
&\quad \textbf{handle } r() \textbf{ with} \\
&\qquad \textbf{return } () \quad\quad\;\; \mapsto \text{cooperate } (rs) \\
&\qquad \langle \text{yield } () \to s \rangle \mapsto \text{cooperate } (rs +\!\!+ [s])
\end{aligned}$$

$$\text{cooperate } [tA, tB] \Longrightarrow ()$$

A1 B1 A2 B2

Example 3: lightweight threads with fork

Effect signature — recursive effect signature

$$\mathsf{Co}\ e = e \,\&\, \{\mathsf{yield} : 1 \twoheadrightarrow 1,\quad \mathsf{fork} : (1 \to 1!\mathsf{Co}\ e) \twoheadrightarrow 1\}$$

Example 3: lightweight threads with fork

Effect signature — recursive effect signature

$$\mathsf{Co}\ e = e\ \&\ \{\mathsf{yield} : 1 \twoheadrightarrow 1,\quad \mathsf{fork} : (1 \to 1!\mathsf{Co}\ e) \twoheadrightarrow 1\}$$

A single cooperative program

$$\begin{aligned}
&\mathsf{main} : 1 \to 1!\mathsf{Co}\ e \\
&\mathsf{main}\,() = \mathsf{print}\ \text{"M1 "}; \mathsf{fork}\,(\lambda().\mathsf{print}\ \text{"A1 "}; \mathsf{yield}\,(); \mathsf{print}\ \text{"A2 "}); \\
&\qquad\qquad \mathsf{print}\ \text{"M2 "}; \mathsf{fork}\,(\lambda().\mathsf{print}\ \text{"B1 "}; \mathsf{yield}\,(); \mathsf{print}\ \text{"B2 "}); \mathsf{print}\ \text{"M3 "}
\end{aligned}$$

Example 3: lightweight threads with fork

Types

$$\text{Thread } e = 1 \to 1!(e \,\&\, \{\text{yield} : 1 \twoheadrightarrow 1\})$$

Handler

$$
\begin{array}{l}
\text{cooperate} : \text{List}\,(\text{Thread } e) \to 1!e \\
\text{cooperate}\,[\,] = (\,) \\
\text{cooperate}\,(r :: rs) = \\
\quad \textbf{handle}\; r(\,)\; \textbf{with} \\
\qquad \textbf{return}\,(\,) \qquad\quad \mapsto \text{cooperate}\,(rs) \\
\qquad \langle \text{yield}\,(\,) \to s \rangle \mapsto \text{cooperate}\,(rs +\!\!+ [s]) \\
\qquad \langle \text{fork}\; t \to s \rangle \quad \mapsto \text{cooperate}\,(t :: rs +\!\!+ [s])
\end{array}
$$

Example 3: lightweight threads with fork

Types

$$\text{Thread } e = 1 \to 1!(e \,\&\, \{\text{yield} : 1 \twoheadrightarrow 1\})$$

Handler

$$
\begin{aligned}
&\text{cooperate} : \text{List}\,(\text{Thread } e) \to 1!e \\
&\text{cooperate}\,[] = () \\
&\text{cooperate}\,(r :: rs) = \\
&\quad \textbf{handle}\; r()\; \textbf{with} \\
&\qquad \textbf{return}\,() \qquad\quad \mapsto \text{cooperate}\,(rs) \\
&\qquad \langle \text{yield}\,() \to s \rangle \mapsto \text{cooperate}\,(rs \mathbin{+\!\!+} [s]) \\
&\qquad \langle \text{fork}\; t \to s \rangle \quad \mapsto \text{cooperate}\,(t :: rs \mathbin{+\!\!+} [s])
\end{aligned}
$$

$$\text{cooperate}\,[\text{main}] \implies ()$$

M1 A1 M2 B1 A2 M3 B2

Example 3: lightweight threads with fork

Types

$$\text{Thread } e = 1 \to 1!(e \,\&\, \{\text{yield} : 1 \twoheadrightarrow 1\})$$

Handler

$$
\begin{aligned}
&\text{cooperate} : \text{List}\,(\text{Thread } e) \to 1!e \\
&\text{cooperate}\,[] = () \\
&\text{cooperate}\,(r :: rs) = \\
&\quad \textbf{handle}\; r()\; \textbf{with} \\
&\qquad \textbf{return}\,() \quad\;\; \mapsto \text{cooperate}\,(rs) \\
&\qquad \langle \text{yield}\,() \to s \rangle \mapsto \text{cooperate}\,(rs +\!\!+ [s]) \\
&\qquad \langle \text{fork}\; t \to s \rangle \;\; \mapsto \text{cooperate}\,(s :: rs +\!\!+ [t])
\end{aligned}
$$

Example 3: lightweight threads with fork

Types

$$\text{Thread } e = 1 \to 1!(e \,\&\, \{\text{yield} : 1 \twoheadrightarrow 1\})$$

Handler

$$
\begin{aligned}
&\text{cooperate} : \text{List}\,(\text{Thread } e) \to 1!e \\
&\text{cooperate}\,[] = () \\
&\text{cooperate}\,(r :: rs) = \\
&\quad \textbf{handle}\; r()\; \textbf{with} \\
&\qquad \textbf{return}\,() \qquad\quad \mapsto \text{cooperate}\,(rs) \\
&\qquad \langle \text{yield}\,() \to s \rangle \mapsto \text{cooperate}\,(rs +\!\!+ [s]) \\
&\qquad \langle \text{fork}\; t \to s \rangle \;\;\; \mapsto \text{cooperate}\,(s :: rs +\!\!+ [t])
\end{aligned}
$$

$$\text{cooperate}\,[\text{main}] \implies ()$$

M1 M2 M3 A1 B1 A2 B2

Built-in effects

Generative state

$$\begin{aligned}\mathsf{GenState} = \{\mathsf{new} &: a.\ a \twoheadrightarrow \mathsf{Ref}\ a,\\ \mathsf{write} &: a.\ (\mathsf{Ref}\ a \times a) \twoheadrightarrow 1,\\ \mathsf{read} &: a.\ \mathsf{Ref}\ a \twoheadrightarrow a\}\end{aligned}$$

Example 4: actors

Process ids

$$\mathsf{Pid}\ a = \mathsf{Ref}\ (\mathsf{List}\ a)$$

Effect signature

$$\begin{aligned}
\mathsf{Actor}\ a = \{\mathsf{self} &: & 1 &\twoheadrightarrow \mathsf{Pid}\ a, \\
\mathsf{spawn} &: b.\ & (1 \to 1!\mathsf{Actor}\ b) &\twoheadrightarrow \mathsf{Pid}\ b, \\
\mathsf{send} &: b.\ & (b \times \mathsf{Pid}\ b) &\twoheadrightarrow 1, \\
\mathsf{recv} &: & 1 &\twoheadrightarrow a\}
\end{aligned}$$

Example 4: actors

Process ids

$$\text{Pid } a = \text{Ref } (\text{List } a)$$

Effect signature

$$\begin{aligned}\text{Actor } a = \{&\text{self} &:& & 1 &\twoheadrightarrow \text{Pid } a,\\ &\text{spawn} &:& b.\ & (1 \to 1!\text{Actor } b) &\twoheadrightarrow \text{Pid } b,\\ &\text{send} &:& b.\ & (b \times \text{Pid } b) &\twoheadrightarrow 1,\\ &\text{recv} &:& & 1 &\twoheadrightarrow a\}\end{aligned}$$

An actor chain

$$\begin{aligned}&\text{spawnMany} : \text{Pid String} \to \text{Int} \to 1!(e \,\&\, \text{Actor String})\\ &\text{spawnMany}\ p\ 0 = \text{send}\ (\text{"ping!"}, p)\\ &\text{spawnMany}\ p\ n = \text{spawnMany}\ (\text{spawn}\ (\lambda().\mathbf{let}\ s = \text{recv}\ ()\ \mathbf{in}\ \text{print}\ \text{"."}; \text{send}\ (s, p)))\ (n-1)\end{aligned}$$

$$\begin{aligned}&\text{chain} : \text{Int} \to 1!(e \,\&\, \text{Actor String} \,\&\, \text{Console})\\ &\text{chain}\ n = \text{spawnMany}\ (\text{self}\ ())\ n; \mathbf{let}\ s = \text{recv}\ ()\ \mathbf{in}\ \text{print}\ s\end{aligned}$$

Example 4: actors — via lightweight threads

$$
\begin{array}{l}
\mathsf{act} : \mathsf{Pid}\ a \to (1 \to 1!(e\ \&\ \mathsf{Actor}\ a)) \to 1!\mathsf{Co}\ (e\ \&\ \mathsf{GenState}) \\
\mathsf{act}\ \mathit{mine}\ t = \\
\quad \textbf{handle}\ t()\ \textbf{with} \\
\qquad \begin{array}{lll}
\textbf{return}\ () & \mapsto & () \\
\langle \mathsf{self}\ () \to r \rangle & \mapsto & \mathsf{act}\ \mathit{mine}\ (\lambda().r\ \mathit{mine}) \\
\langle \mathsf{spawn}\ \mathit{you} \to r \rangle & \mapsto & \textbf{let}\ \mathit{yours} = \mathsf{new}\ []\ \textbf{in} \\
 & & \mathsf{fork}\ (\lambda().\mathsf{act}\ \mathit{yours}\ (\mathit{you}\ ()));\ \mathsf{act}\ \mathit{mine}\ (\lambda().r\ \mathit{yours}) \\
 & & r\ \mathit{mine}\ \mathit{yours} \\
\langle \mathsf{send}\ (m, \mathit{yours}) \to r \rangle & \mapsto & \textbf{let}\ \mathit{ms} = \mathsf{read}\ \mathit{yours}\ \textbf{in} \\
 & & \mathsf{write}\ (\mathit{yours}, \mathit{ms} +\!\!+ [m]);\ \mathsf{act}\ \mathit{mine}\ r \\
\langle \mathsf{recv}\ () \to r \rangle & \mapsto & \textbf{case}\ \mathsf{read}\ \mathit{mine}\ \textbf{of} \\
 & & \quad \begin{array}{lll}
 [] & \mapsto & \mathsf{yield}\ ();\ \mathsf{act}\ \mathit{mine}\ (\lambda().r\ (\mathsf{recv}\ ())) \\
 (m :: \mathit{ms}) & \mapsto & \mathsf{write}\ (\mathit{mine}, \mathit{ms});\ \mathsf{act}\ \mathit{mine}\ (\lambda().r\ m)
 \end{array}
\end{array}
\end{array}
$$

Example 4: actors — via lightweight threads

$$\begin{array}{l}
\mathsf{act} : \mathsf{Pid}\ a \to (1 \to 1!(e\ \&\ \mathsf{Actor}\ a)) \to 1!\mathsf{Co}\ (e\ \&\ \mathsf{GenState}) \\
\mathsf{act}\ mine\ t = \\
\quad \mathbf{handle}\ t()\ \mathbf{with} \\
\qquad \begin{array}{lll}
\mathbf{return}\ () & \mapsto & () \\
\langle \mathsf{self}\ () \to r \rangle & \mapsto & \mathsf{act}\ mine\ (\lambda().r\ mine) \\
\langle \mathsf{spawn}\ you \to r \rangle & \mapsto & \mathbf{let}\ yours = \mathsf{new}\ []\ \mathbf{in} \\
 & & \mathsf{fork}\ (\lambda().\mathsf{act}\ yours\ (you\ ()));\ \mathsf{act}\ mine\ (\lambda().r\ yours) \\
 & & r\ mine\ yours \\
\langle \mathsf{send}\ (m, yours) \to r \rangle & \mapsto & \mathbf{let}\ ms = \mathsf{read}\ yours\ \mathbf{in} \\
 & & \mathsf{write}\ (yours, ms \mathbin{+\!\!+} [m]);\ \mathsf{act}\ mine\ r \\
\langle \mathsf{recv}\ () \to r \rangle & \mapsto & \mathbf{case}\ \mathsf{read}\ mine\ \mathbf{of} \\
 & & \quad \begin{array}{lll}
 [] & \mapsto & \mathsf{yield}\ ();\ \mathsf{act}\ mine\ (\lambda().r\ (\mathsf{recv}\ ())) \\
 (m :: ms) & \mapsto & \mathsf{write}\ (mine, ms);\ \mathsf{act}\ mine\ (\lambda().r\ m)
 \end{array}
\end{array}
\end{array}$$

$$\mathsf{cooperate}\ [\lambda().\mathsf{act}\ (\mathsf{new}\ [])\ (\lambda().\mathsf{chain}\ 64)] \implies ()$$

....ping!

Other use-cases

- reactive programming
- dependency injection
- mocking
- fuzzing
- automatic differentiation
- probabilistic programming
- backtracking

Example 5: lightweight threads in C++

```cpp
struct Yield : eff::command<> { };
struct Fork : eff::command<> {
  std::function<void()> proc;
};

void yield() {
  eff::invoke_command(Yield{});
}
void fork(std::function<void()> proc) {
  eff::invoke_command(Fork{{}, proc});
}

void mainThread() {
  std::cout << "M1 "; fork([=]() {std::cout << "A1 "; yield(); std::cout << "A2 "});
  std::cout << "M2 "; fork([=]() {std::cout << "B1 "; yield(); std::cout << "B2 "});
  std::cout << "M3 ";
}
```

Example 5: lightweight threads in C++

```
using Res = eff::resumption<void()>;
class Scheduler : public eff::handler<void, void, Yield, Fork> {
public:
  static void Start(std::function<void()> f) {
    queue.push_back(eff::wrap<Scheduler>(f));
    while (!queue.empty()) {
      Res resumption = std::move(queue.front());
      queue.pop_front();
      std::move(resumption).resume();
    }
  }
private:
  static std::list<Res> queue;
  void handle_command(Yield, Res r) override {
    queue.push_back(std::move(r));
  }
  void handle_command(Fork f, Res r) override {
    queue.push_back(std::move(r));
    queue.push_back(eff::wrap<Scheduler>(f.proc));
  }
  void handle_return() override { }
};
```

Example 5: lightweight threads in C++

```
int main() {
  Scheduler::Start(mainThread);
}
```

Example 5: lightweight threads in C++

```
int main() {
  Scheduler::Start(mainThread);
}
```

M1 A1 M2 B1 A2 M3 B2

Effect handler oriented programming languages

Eff	https://www.eff-lang.org/
Effekt	https://effekt-lang.org/
Frank	https://github.com/frank-lang/frank
Helium	https://bitbucket.org/pl-uwr/helium
Links	https://www.links-lang.org/
Koka	https://github.com/koka-lang/koka
OCaml 5	https://github.com/ocamllabs/ocaml-multicore/wiki

Timeline

Short term

- One-shot effect handlers / delimited continuations without effect types (OCaml 5, Java 19, Wasm stack switching)
- Primary users: compiler engineers, low-level library developers
- Largely hidden from application developers (e.g. Java 19's virtual threads, OCaml 5's EIO library, Daan Leijen's Node.C library)

Longer term

- Effect type systems to support more robust programming in the large
- Potential compromises for legacy systems based on capability-passing style and modal types
- Efficient compilation of deeply-nested handlers
- Multishot effect handlers for backtracking, probabilistic programming, etc.
- Combination with linear/affine type systems (e.g. languages like Rust)

Resources

The EHOP project website
https://effect-handlers.org/

Jeremy Yallop's effects bibliography
https://github.com/yallop/effects-bibliography

Daniel Hillerström's PhD thesis
Foundations for programming and implementing effect handlers
Hillerström (The University of Edinburgh, 2022)

OCaml 5 effect handlers
Retrofitting effect handlers to OCaml
Sivaramakrishnan, Dolan, White, Jaffer, Kelly, Madhavapeddy (PLDI 2021)

C++ effects library
High-level effect handlers in C++
Ghica, Lindley, Bravo, Piróg (OOPSLA 2022)