Handlers for Non-Monadic Computations

Ruben Pieters \(^1\)  
Tom Schrijvers \(^1\)  
Exequiel Rivas \(^2\)

September 8, 2018
Agenda

- Introduction to Effect Handlers
- Monadic Effect Handlers
- Motivation Non-Monadic Effect Handlers
- Wrap-up
What are (effect) handlers?

Represent occurrence of side-effects with calls to operations. The meaning of these operations are given by handlers.

\(^1\text{http://www.eff-lang.org}\)
What are (effect) handlers?

Represent occurrence of side-effects with calls to operations. The meaning of these operations are given by handlers.

Various implementations

- as language (Eff\(^1\), Frank, Koka, . . . )
- as library (e.g. in Haskell)

\(^1\)http://www.eff-lang.org
Throw

Operation

\[
\text{throw} : \text{String} \rightarrow \text{a}
\]

Computation

\[
\text{let div } x \ y = \\
\text{if } y \neq 0 \\
\text{then } x / y \\
\text{else throw "division by zero!"}
\]
Handling Throw

Evaluation

\texttt{div 10 0}
Handling Throw

Evaluation

\[
\text{div } 10 \ 0 \\
= \text{if } 0 \neq 0 \\
\quad \text{then } 10 \ / \ 0 \\
\quad \text{else throw } \text{"division by zero!"}
\]
Handling Throw

Evaluation

div 10 0
= if false
    then 10 / 0
else throw "division by zero!"
Handling Throw

Evaluation

\[
\text{div 10 0 } \\
= \text{ if false } \\
\quad \text{then 10 / 0 } \\
\quad \text{else throw "division by zero!" } \\
= \text{ throw "division by zero!"}
\]
Handling Throw

Evaluation

\[
\begin{align*}
div \ 10 \ 0 & \\
= \ & \text{if} \ \text{false} \\
& \quad \text{then} \ 10 \ / \ 0 \\
& \quad \text{else} \ \text{throw} \ "\text{division by zero!}" \\
= & \ \text{throw} \ "\text{division by zero!}" \\
> & \ \text{uncaught operation: throw}
\end{align*}
\]
Handling Throw

**Handler**

```ocaml
let h = handler
val x -> x
throw s k -> 0
```

**Evaluation**

```ocaml
handle div 10 0 with h
```
Handling Throw

**Handler**

```plaintext
let h = handler
  val x -> x
  throw s k -> 0
```

**Evaluation**

```plaintext
handle div 10 0 with h
```
Handling Throw

**Handler**

```ocaml
let h = handler
val x -> x
throw s k -> 0
```

**Evaluation**

```ocaml
handle div 10 0 with h
```
Handling Throw

**Handler**

```plaintext
let h = handler
val x  -> x
throw s k -> 0
```

**Evaluation**

```plaintext
handle div 10 0 with h
= handle
  if 0 != 0
  then 10 / 0
  else throw "division by zero!"
with h
```
Handling Throw

**Handler**

```ocaml
let h = handler
  val x -> x
  throw s k -> 0
```

**Evaluation**

```ocaml
handle div 10 0 with h
= handle
  if false
  then 10 / 0
  else throw "division by zero!"
with h
```
Handling Throw

**Handler**

```plaintext
let h = handler
  val x -> x
  throw s k -> 0
```

**Evaluation**

```plaintext
handle div 10 0 with h
= handle
  throw "division by zero!"
with h
```
Handling Throw

**Handler**

```
let h = handler
val x   -> x
throw s k  -> 0
```

**Evaluation**

```
handle div 10 0 with h
= handle
  throw "division by zero!"
with h
> 0 : int
```
Handling Throw - Value Case

### Handler

```haskell
let h = handler
  val x -> x
  throw s k -> 0
```

### Evaluation

```haskell
handle div 10 5 with h
```
Handling Throw - Value Case

**Handler**

```plaintext
let h = handler
val x -> x
throw s k -> 0
```

**Evaluation**

```plaintext
handle div 10 5 with h
= handle
  if 5 != 0
    then 10 / 5
  else throw "division by zero!"
with h
```
Handling Throw - Value Case

**Handler**

```plaintext
let h = handler
val x    -> x
throw s k -> 0
```

**Evaluation**

```plaintext
handle div 10 5 with h
= handle
  if true
    then 10 / 5
  else throw "division by zero!"
with h
```
Handling Throw - Value Case

**Handler**

```ocaml
define h = handler
  val x -> x
  throw s k -> 0
```

**Evaluation**

```ocaml
handle div 10 5 with h
= handle
  10 / 5
with h
```
Handling Throw - Value Case

**Handler**

```ocaml
let h = handler
val x -> x
throw s k -> 0
```

**Evaluation**

```ocaml
handle div 10 5 with h
= handle
  2
  with h
```
Handling Throw - Value Case

**Handler**

```plaintext
let h = handler
  val x -> x
  throw s k -> 0
```

**Evaluation**

```plaintext
handle div 10 5 with h
= handle
  2
  with h
> 2 : int
```
Handling Throw

**Handler**

```plaintext
let h = handler
  val x    -> Some x
  throw s k -> None
```
Handling Throw

**Handler**

```ocaml
let h = handler
  val x     -> Some x
throw s k -> None
```

**Evaluation**

```ocaml
handle div 10 0 with h
> None : int option
```
# Handling Throw

## Handler

```
let h = handler
    val x       -> Some x
    throw s k  -> None
```

## Evaluation

```
handle div 10 0 with h
> None : int option
handle div 10 5 with h
> Some 2 : int option
```
Special K

**Handler**

```latex
let h = handler
val x -> x
throw s k -> 0
```

The $k$ variable gives access to the continuation.
What is a Continuation?

Small Example

⋄ + 1
What is a Continuation?

Small Example

\( + 1 \)

: int -> int
What is a Continuation?

Small Example

◊ + 1
: int -> int

Operations introduce ◊ and we access the continuation with k
Handling with Continuation

Without k

```plaintext
handle throw "" + 1 with
  throw s k -> 42
```

With k

```plaintext
handle throw "" + 1 with
  throw s k -> k 42
```
Handling with Continuation

Without k

```plaintext
handle throw "" + 1 with
  throw s k -> 42
= 42 where
  k ← ⋄ + 1
  s ← ""
```

With k

```plaintext
handle throw "" + 1 with
  throw s k -> k 42
```
Handling with Continuation

**Without k**

```plaintext
handle throw "" + 1 with
  throw s k -> 42
= 42 where
  k ← ⋄ + 1
  s ← ""
> 42
```

**With k**

```plaintext
handle throw "" + 1 with
  throw s k -> k 42
```
Handling with Continuation

**Without k**

```plaintext
handle throw "" + 1 with
  throw s k -> 42
= 42 where
  k ← ◇ + 1
  s ← ""
> 42
```

**With k**

```plaintext
handle throw "" + 1 with
  throw s k -> k 42
= k 42 where
  k ← ◇ + 1
  s ← ""
```
Handling with Continuation

Without k

```
handle throw "" + 1 with
  throw s k -> 42
= 42 where
  k ← ⋄ + 1
  s ← ""
> 42
```

With k

```
handle throw "" + 1 with
  throw s k -> k 42
= k 42 where
  k ← ⋄ + 1
  s ← ""
> 43
```
Handling Rules

Value Rule

\[
\text{handle } v \text{ with } \text{val } x \rightarrow c_v
\]

\[
= c_v \text{ where } x \leftarrow v
\]
Handling Rules

Value Rule

\[
\text{handle } v \text{ with val } x \rightarrow c_v
\]
\[
= c_v \text{ where }
\]
\[
x \leftarrow v
\]

Operation Rule

\[
\text{handle } K(op_i \ p') \text{ with } op_i \ p \ k \rightarrow c_i
\]
\[
= c_i \text{ where }
\]
\[
p \leftarrow p'
\]
\[
k \leftarrow \backslash n \rightarrow \text{handle } (K \ n)
\]
State

Operation

\[
\begin{align*}
\text{get} & : () \rightarrow s \\
\text{put} & : s \rightarrow ()
\end{align*}
\]
State

Computation

let comp =
  let s = get () in
  set s ;
  s
State

Different Interpretations

State Passing Functions (State Monad):

\[
\text{handle \ comp\ with\ h} : s \rightarrow (a, s)
\]
Different Interpretations

State Passing Functions (State Monad):

\[
\text{handle comp with } h \\
: s \rightarrow (a, s)
\]

Connect to DB with IO:

\[
\text{handle comp with } h \\
: \text{IO } a
\]
State

Different Interpretations

State Passing Functions (State Monad):

\[
\text{handle \ comp \ with \ h} \\
: \ s \to (a, s)
\]

Connect to DB with IO:

\[
\text{handle \ comp \ with \ h} \\
: \ IO \ a
\]

...
Counting Operations?

Computation

```plaintext
let comp =
  let s = get () in
  set s ;
s
```

```plaintext
let comp =
  let s = get () in
  set s ;
s
```
Counting Operations?

**Handler**

```
let h = handler
```
Counting Operations?

Handler

```ocaml
let h = handler
val x  -> 0
```
Counting Operations?

**Handler**

```ocaml
let h = handler
val x -> 0
put p k -> (k ()) + 1
```
Counting Operations?

**Handler**

```ocaml
define handler
  val x -> 0
  put p k -> (k ()) + 1
  get p k -> (k ?) + 1
```
Counting Operations?

**Handler**

```ocaml
define h = handler
  let x = 0
  let put p k = (k () + 1)
  let get p k = (k X + 1)
```

```
Monadic Handlers

Computation

```haskell
let comp’ =
  let s = get () in
  for i in s: set i ;
  s
```
Monadic Handlers

Computation

```plaintext
let comp' =
  let s = get () in
  for i in s: set i ;
  s

Monadic handlers must be able to handle all monadic computations.
This condition restricts the possibilities of these handlers.
```
Inspiration

Algebraic Effects and Effect Handlers for Idioms and Arrows

_Sam Lindley_

Introduces calculus $\lambda_{flow}$ which defines handler constructs for applicative, arrow and monad effects

By _restricting the computations_, we _increase the possibilities_ for the corresponding handlers.
Category Theoretical Approach

Based on:

Notions of Computations as Monoids

*Exequiel Rivas and Mauro Jaskelliof*

Monads, Applicatives and Arrows are monoids in monoidal categories
Category Theoretical Approach

Based on:

Notions of Computations as Monoids

*Exequiel Rivas and Mauro Jaskelioff*

Monads, Applicatives and Arrows are monoids in monoidal categories

All these notions are **monoids** in a monoidal category, can we **derive a notion of handler** from this?
### The Table

<table>
<thead>
<tr>
<th>Handler</th>
<th>Free Algebra</th>
<th>Free Monoid</th>
<th>Free Monoid (⁻)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f : A \rightarrow B$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Handling Rules**

[val x -> c_v]
The Table

| Handler | Free Algebra | Free Monoid | Free Monoid (|-) |
|---------|-------------|-------------|-----------------|
|         | $f : A \to B$ |             |                 |
|         | $b : \Sigma B \to B$ |             |                 |

Handling

Handling Rules

$op_i \ p \ k \to c_i^2$

$^2\Sigma = P_0 \times -N_0 + \ldots + P_n \times -N_n$
## The Table

<table>
<thead>
<tr>
<th>Handler</th>
<th>Free Algebra</th>
<th>Free Monoid</th>
<th>Free Monoid ((\bot))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f : A \rightarrow B)</td>
<td>(b : \Sigma B \rightarrow B)</td>
<td>(h : \Sigma^* A \rightarrow B)</td>
<td></td>
</tr>
</tbody>
</table>

### Handling Rules

\[h \circ \langle \Sigma \circ f \rangle = b \circ \Sigma\]

\[h \circ \langle \Sigma \circ e \rangle = e_M\]

\[h \circ \langle \Sigma \circ m \rangle = \phi_2 \circ (\Sigma \otimes h)\]

\[m_M \circ (h \otimes h)\]

The table outlines the relationships between handlers and their respective algebraic structures. The `\(\Sigma\)` denotes a sum type, and `\(\otimes\)` denotes a tensor product. The `\(\Sigma^*\)` represents the Kleene star operation, which is a common notation in formal language theory for representing Kleene algebras.
## The Table

| Handler | Free Algebra | Free Monoid | Free Monoid (|−|) |
|---------|--------------|-------------|-----------------|
| $f : A \rightarrow B$ | $b : \Sigma B \rightarrow B$ | | |
| $h : \Sigma^* A \rightarrow B$ | | | |

**Handling Rules**

$h \circ v = f$

**handle v with**

```
(val x -> c_v) = c_v
```

where

```
x <- v
```
### The Table

<table>
<thead>
<tr>
<th>Handler</th>
<th>Free Algebra</th>
<th>Free Monoid</th>
<th>Free Monoid ((\square))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f : A \rightarrow B)</td>
<td>(b : \Sigma B \rightarrow B)</td>
<td>(\phi_1 : I \rightarrow F)</td>
<td></td>
</tr>
<tr>
<td>(h : \Sigma^* A \rightarrow B)</td>
<td>(h \circ v = f)</td>
<td>(h \circ op = b \circ \Sigma h)</td>
<td></td>
</tr>
</tbody>
</table>

handle \(K(op_i \; p')\) with \((op_i \; p \; k \rightarrow c_i) = c_i\)

where

- \(p \leftarrow p'\)
- \(k \leftarrow n \rightarrow \text{handle } (K \; n)\)
# The Table

| Handler | Free Algebra | Free Monoid | Free Monoid (|−|) |
|---------|-------------|-------------|----------------|
| Handler | f : A → B   | b : ΣB → B  |                |
| Handling| h : Σ*A → B |             |                |
| Handling Rules | h ◦ v = f | h ◦ op = b ◦ Σh |                |
## The Table

<table>
<thead>
<tr>
<th>Handler</th>
<th>Free Algebra</th>
<th>Free Monoid</th>
<th>Free Monoid $(\cdot \cdot)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f : A \to B$</td>
<td></td>
<td>$f : \Sigma \to M$</td>
<td>$(M, e_M, m_M)$</td>
</tr>
<tr>
<td>$b : \Sigma B \to B$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h : \Sigma^* A \to B$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handling Rules</td>
<td>$h \circ v = f$</td>
<td>$h \circ \text{op} = b \circ \Sigma h$</td>
<td></td>
</tr>
</tbody>
</table>
## The Table

<table>
<thead>
<tr>
<th>Handler</th>
<th>Free Algebra</th>
<th>Free Monoid</th>
<th>Free Monoid ((-))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f : A \rightarrow B)</td>
<td>(f : \Sigma \rightarrow M)</td>
<td>((M, e_M, m_M))</td>
<td></td>
</tr>
<tr>
<td>(b : \Sigma B \rightarrow B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(h : \Sigma^* A \rightarrow B)</td>
<td>(h : \Sigma^* \rightarrow M)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handling Rules</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(h \circ v = f)</td>
<td>(h \circ \text{op} = b \circ \Sigma h)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## The Table

<table>
<thead>
<tr>
<th>Handler</th>
<th>Free Algebra</th>
<th>Free Monoid</th>
<th>Free Monoid ((-))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f : A \rightarrow B)</td>
<td>(b : \Sigma B \rightarrow B)</td>
<td>(f : \Sigma \rightarrow M)</td>
<td>((M, e_M, m_M))</td>
</tr>
<tr>
<td>(h : \Sigma^* A \rightarrow B)</td>
<td>(h : \Sigma^* \rightarrow M)</td>
<td>(h \circ \text{ins} = f)</td>
<td></td>
</tr>
<tr>
<td>(h \circ \text{op} = b \circ \Sigma h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handler</td>
<td>Free Algebra</td>
<td>Free Monoid</td>
<td>Free Monoid $\langle - \rangle$</td>
</tr>
<tr>
<td>---------</td>
<td>--------------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>$f : A \to B$</td>
<td>$f : \Sigma \to M$</td>
<td>$(M, e_M, m_M)$</td>
<td></td>
</tr>
<tr>
<td>$b : \Sigma B \to B$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h : \Sigma^* A \to B$</td>
<td>$h : \Sigma^* \to M$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handling Rules</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h \circ v = f$</td>
<td>$h \circ \text{ins} = f$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h \circ \text{op} = b \circ \Sigma h$</td>
<td>$h \circ e_{\Sigma^*} = e_M$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$h \circ m_{\Sigma^*} =$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$m_M \circ (h \otimes h)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## The Table

<table>
<thead>
<tr>
<th>Handler</th>
<th>Free Algebra</th>
<th>Free Monoid</th>
<th>Free Monoid ((\cdot))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(f : A \to B)</td>
<td>(f : \Sigma \to M)</td>
<td>(\phi_1 : I \to F)</td>
</tr>
<tr>
<td></td>
<td>(b : \Sigma B \to B)</td>
<td>((M, e_M, m_M))</td>
<td>(\phi_2 : \Sigma \otimes F \to F)</td>
</tr>
<tr>
<td>Handling</td>
<td>(h : \Sigma^* A \to B)</td>
<td>(h : \Sigma^* \to M)</td>
<td></td>
</tr>
<tr>
<td>Handling Rules</td>
<td>(h \circ \nu = f)</td>
<td>(h \circ \text{ins} = f)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(h \circ \text{op} = b \circ \Sigma h)</td>
<td>(h \circ e_{\Sigma^*} = e_M)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(h \circ m_{\Sigma^*} = m_M \circ (h \otimes h))</td>
<td></td>
</tr>
</tbody>
</table>
# The Table

<table>
<thead>
<tr>
<th>Handler</th>
<th>Free Algebra</th>
<th>Free Monoid</th>
<th>Free Monoid (⊥–)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$f : A \rightarrow B$</td>
<td>$f : \Sigma \rightarrow M$</td>
<td>$\phi_1 : I \rightarrow F$</td>
</tr>
<tr>
<td></td>
<td>$b : \Sigma B \rightarrow B$</td>
<td>$(M, e_M, m_M)$</td>
<td>$\phi_2 : \Sigma \otimes F \rightarrow F$</td>
</tr>
<tr>
<td>Handling</td>
<td>$h : \Sigma^* A \rightarrow B$</td>
<td>$h : \Sigma^* \rightarrow M$</td>
<td>$h : \Sigma^* \rightarrow F$</td>
</tr>
<tr>
<td>Handling Rules</td>
<td>$h \circ \nu = f$</td>
<td>$h \circ \text{ins} = f$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$h \circ \text{op} = b \circ \Sigma h$</td>
<td>$h \circ e_{\Sigma^*} = e_M$</td>
<td>$h \circ m_{\Sigma^*} =$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$m_M \circ (h \otimes h)$</td>
<td></td>
</tr>
</tbody>
</table>
## The Table

<table>
<thead>
<tr>
<th>Handler</th>
<th>Free Algebra</th>
<th>Free Monoid</th>
<th>Free Monoid $(\neg \cdot)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f : A \to B$</td>
<td>$f : \Sigma \to M$</td>
<td>$\phi_1 : I \to F$</td>
<td></td>
</tr>
<tr>
<td>$b : \Sigma B \to B$</td>
<td>$(M, e_M, m_M)$</td>
<td>$\phi_2 : \Sigma \otimes F \to F$</td>
<td></td>
</tr>
<tr>
<td>$h : \Sigma^* A \to B$</td>
<td>$h : \Sigma^* \to M$</td>
<td>$h : \Sigma^* \to F$</td>
<td></td>
</tr>
<tr>
<td>$h \circ \nu = f$</td>
<td>$h \circ \text{ins} = f$</td>
<td>$h \circ \text{in}_1 = \phi_1$</td>
<td></td>
</tr>
<tr>
<td>$h \circ \text{op} = b \circ \Sigma h$</td>
<td>$h \circ e_{\Sigma^*} = e_M$</td>
<td>$h \circ \text{in}_2 =$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$h \circ m_{\Sigma^*} =$</td>
<td>$\phi_2 \circ (\Sigma \otimes h)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$m_M \circ (h \otimes h)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## The Table

<table>
<thead>
<tr>
<th>Handler</th>
<th>Free Algebra</th>
<th>Free Monoid</th>
<th>Free Monad $\langle \dashv \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f : A \to B$</td>
<td>$f : \Sigma \to M$</td>
<td>$\phi_1 : A \to FA$</td>
<td></td>
</tr>
<tr>
<td>$b : \Sigma B \to B$</td>
<td>$(M, e_M, m_M)$</td>
<td>$\phi_2 : \Sigma(FA) \to FA$</td>
<td></td>
</tr>
<tr>
<td>$h : \Sigma^* A \to B$</td>
<td>$h : \Sigma^* \to M$</td>
<td>$h : \Sigma^* A \to FA$</td>
<td></td>
</tr>
<tr>
<td>$h \circ v = f$</td>
<td>$h \circ ins = f$</td>
<td>$h \circ in_1 = \phi_1$</td>
<td></td>
</tr>
<tr>
<td>$h \circ op = b \circ \Sigma h$</td>
<td>$h \circ e_{\Sigma^*} = e_M$</td>
<td>$h \circ in_2 = \phi_2 \circ \Sigma h$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$h \circ m_{\Sigma^*} = m_M \circ (h \otimes h)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Counting Operations

### Handler

```
let h = handler

val x -> Δ_N 0
put p y k f -> k + 1
get p y k f -> k + 1
```

$$I \rightarrow F$$

$$\phi_1$$
Counting Operations

**Handler**

```ocaml
let h = handler
val x -> Δₜₜ 0
put p y k f -> k + 1
get p y k f -> k + 1
```

\[Id(A) \rightarrow FA\]
Counting Operations

**Handler**

```plaintext
let h = handler
val x -> Δ_N 0
put p y k f -> k + 1
get p y k f -> k + 1
```

\[ A \rightarrow FA \]
Counting Operations

**Handler**

```ocaml
let h = handler
val x -> Δₙ 0
put p y k f -> k + 1
get p y k f -> k + 1
```

\[ \Sigma \otimes F \rightarrow F \quad \text{(} \phi_2 \text{)} \]
Counting Operations

Handler

```ocaml
let h = handler
  val x -> ΔN 0
  put p y k f -> k + 1
  get p y k f -> k + 1

Σ ⋆ F → F
```
Counting Operations

**Handler**

```plaintext
let h = handler
val x -> Δ_N 0
put p y k f -> k + 1
get p y k f -> k + 1
```

\[(P_i \times -^N_i) \ast F \rightarrow FA\]
Counting Operations

**Handler**

```ml
let h = handler
val x -> Δ₀
put p y k f -> k + 1
get p y k f -> k + 1
```

```math
\int^{Y,Z} (P_i \times Y^{N_i} \times FZ \times (Y \times Z \to -)) \to F
```
Counting Operations

**Handler**

\[
\text{let } h = \text{handler}
\]

\[
\begin{align*}
\text{val } x & \to \Delta_{\mathbb{N}} 0 \\
\text{put } p \ y \ k \ f & \to k + 1 \\
\text{get } p \ y \ k \ f & \to k + 1
\end{align*}
\]

\[
\int^{Y, Z} (P_i \times Y^{N_i} \times FZ \times (Y \times Z \to A)) \to FA
\]
Key Point

The standard handlers can be derived from Free Algebras, while our non-monadic handlers are derived from Free Monoids.
Paper

- Much more in-depth theory
- Using less expressive handlers on more expressive computations (e.g. monadic handler on applicative computation, by utilizing lax monoidal functors)
Ongoing Work

- Simplify signatures
- Investigate use of Continuation Monad to obtain interface in the base category $\mathcal{C}$
Thanks for your attention!

ruben.pieters@cs.kuleuven.be