Towards efficient implementations of effect handlers

Steven Keuchel

with Tom Schrijvers
Disclaimer
Disclaimer

Investigative Talk
Disclaimer

Investigative Talk

Don’t expect solutions from me
Don’t expect solutions from me

I expect solutions from you.
The ideal imperative language
Ideal imperative
Ideal imperative

Compositionality

Abstraction

Ease of use
Ideal imperative

Compositionality

Abstraction

Ease of use

Monads

Monad transformers

Monad classes
Ideal imperative

Compositionality

Abstraction

Ease of use

Monads

Monad transformers

Monad classes

Interaction
Ideal imperative

Compositionality

Abstraction

Ease of use

Monads

Monad transformers

Monad classes

Interaction 😞
Effect handlers
Effect handlers

Edwin Brady: “Programming and reasoning with algebraic effects and dependent types.”, ICFP’13.


Andrej Bauer and Matija Pretnar: “Programming with algebraic effects and handlers.”, JLAMP’14.


**Exception Handlers**

```plaintext
catch
  (if (n == 0)
    then throw DivZero
    else m `div` n)
```

```plaintext
(\DivZero -> 0)
```

**Effect Handlers**

```plaintext
handle
  (if (x == 0)
    then throw DivZero
    else m `div` n)
```

```plaintext
with
  [throw DivZero ? k] -> 0
  [x] -> x
```
Exception Handlers

```
catch (if (n == 0)
    then throw DivZero
   else m `div` n)

(
`DivZero` -> 0)
```

Effect Handlers

```
handle (if (x == 0)
    then throw DivZero
   else m `div` n)

with [throw DivZero ? k] -> 0
    [x] -> x
```

Operation
Exception Handlers

```
catch (if (n == 0)
    then throw DivZero
  else m 'div' n)
  neurop (DivZero -> 0)
```

Effect Handlers

```
handle (if (x == 0)
    then throw DivZero
  else m 'div' n)
with [throw DivZero ? k] -> 0
[x] -> x
```

Operation

Continuation
Exception effect handler
Exception effect handler

1. Abstract effect operations

\[
\text{sig } \text{Exception } e \\
\quad = \ \text{throw } e :: ()
\]
Exception effect handler

1. Abstract effect operations

\[
\text{sig}\ Exception\ e \\
\quad =\ \text{throw}\ e::() \\
\]

2. Handler implementation

\[
\text{either}\ ::\ [\text{Exception}\ e]\ a\ \rightarrow\ \text{Either}\ e\ a \\
\text{either}\ [\text{throw}\ e\ ?\ k]\ =\ \text{Left}\ e \\
\text{either}\ [a]\ =\ \text{Right}\ a \\
\]

\[
\text{maybe}\ ::\ [\text{Exception}\ e]\ a\ \rightarrow\ \text{Maybe}\ a \\
\text{maybe}\ [\text{throw}\ e\ ?\ k]\ =\ \text{Nothing} \\
\text{maybe}\ [a]\ =\ \text{Just}\ a
\]
State effect handler
State effect handler

1. Abstract effect operations

\[
\text{sig State } s
\]

\[
= \text{get } :: s
\]

\[
= \text{put } s :: ()
\]
State effect handler

1. Abstract effect operations

\[
\text{sig State } s \\
= \text{get} :: s \\
= \text{put } s :: ()
\]

2. Handler implementation

\[
\text{state } :: \ s \rightarrow [\text{State } s] \ a \rightarrow (a,s) \\
\text{state } s \ [\text{get } \ k] = \text{state } s \ ? \ k \ s \\
\text{state } _ \ [\text{put } s \ ? \ k] = \text{state } s \ ? \ k \ () \\
\text{state } s \ [a] = (a,s)
\]
Ideal imperative  Handlers

Compositionality

Abstraction

Ease of use
Ideal imperative

Compositionality

Abstraction

Ease of use

Handlers

Magic / Math

Signatures

Continuations
Ideal imperative

- Compositionality
- Abstraction
- Ease of use

Handlers

- Magic / Math
- Signatures
- Continuations
Ideal imperative

Compositionality

Abstraction

Ease of use

Handlers

Magic / Math

Signatures

Continuations

EASY
We are convinced!
Our goal
World domination!
Plan to Take Over the World:
Open issues

- Effect Polym.
- Effect Typing
- Effect Inference
- Semantics
- Reasoning
One thing overlooked
Performance :(  
Haskell Benchmark

\[
\text{loop} = \text{do}
\begin{align*}
  &s \leftarrow \text{get} \\
  &\text{if } s > 0 \\
  &\quad \text{then } \text{put } (s-1) \gg \text{loop} \\
  &\quad \text{else } \text{return } s
\end{align*}
\]
Haskell Benchmark
Haskell Benchmark

![Bar chart showing comparison between MTL and HIA]

- MTL
- HIA

One
Haskell Benchmark

![Haskell Benchmark Chart]

- One
- Two
- Three
- Four
- Five

- MTL
- HIA
- Frankly
- Extensible Effects
Haskell Benchmark

- One
- Two
- Three
- Four
- Five

MTL
HIA
hProlog Benchmark

- **Base**: 0
- **One**: 30,000
- **Two**: 60,000
- **Three**: 90,000
- **Four**: 120,000
- **Five**: 120,000

Bar chart showing the performance of different stages in the hProlog Benchmark.
Why are handlers so slow?
Free Monads
Free monads
Free monads

- put
- choose
  - throw
  - put
  - write
- return
Delimited control
Delimited continuations

```
loop = do
  s <- get
  if s > 0
    then put (s-1) >> loop
  else return s
```
Delimited continuations

\[
\text{loop} = \text{do}
\]

\[
\begin{align*}
\text{s} & \gets \text{get} \\
\text{if } s & \gt 0 \\
\text{then } & \text{put } (s-1) \gg \text{loop} \\
\text{else } & \text{return } s
\end{align*}
\]
Delimited continuations

\[
\begin{align*}
\text{loop} &= \text{do} \\
&\quad \text{s} \leftarrow \text{get} \\
&\quad \text{if } s > 0 \\
&\quad \quad \text{then put } (s-1) >> \text{loop} \\
&\quad \text{else return } s
\end{align*}
\]
Delimited continuations

\[\text{loop} = \text{do} \]
\[s \leftarrow \text{get}\]
\[\text{if } s > 0 \text{ then put } (s-1) \gg \text{loop} \]
\[\text{else return } s\]
Delimited continuations

\[
\text{loop} = \text{do} \\
\text{s \xleftarrow{\text{get}}} \\
\text{if } s > 0 \text{ then put } (s-1) \gg \text{loop} \\
\text{else return } s
\]
Delimited continuations

\[ \text{loop} = \text{do} \]
\[ \begin{align*}
  &s \leftarrow \text{get} \\
  \text{if } s > 0 &\text{ then put } (s-1) \gg \text{loop} \\
  \text{else } &\text{return } s
\end{align*} \]
Delimited continuations

loop = do
  s <- get
  if s > 0
    then put (s-1) >> loop
  else return s
Delimited continuations

```
loop = do
  s <- get
  if s > 0
    then put (s-1) >> loop
  else return s
```
Delimited continuations

loop = do
  s <- get
  if s > 0
    then put (s-1) >> loop
    else return s
Delimited continuations

\[
\text{loop} = \text{do} \\
\text{s} \leftarrow \text{get} \\
\text{if } s > 0 \\
\text{then put } (s-1) \gg \text{loop} \\
\text{else return } s
\]
Delimited continuations

\[
\text{loop} = \text{do} \\
\quad s \leftarrow \text{get} \\
\quad \text{if } s > 0 \\
\qquad \text{then } \text{put } (s-1) \gg \text{loop} \\
\quad \text{else } \text{return } s
\]
Delimited continuations

\[
\text{loop} = \text{do}
\]

\[
\begin{align*}
\text{s} & \leftarrow \text{get} \\
\text{if} \ s > 0 \\
\text{then} \ & \text{put (s-1)} \gg \text{loop} \\
\text{else} \ & \text{return} \ s
\end{align*}
\]
Delimited continuations

\[
\text{loop} = \text{do}
\begin{align*}
  s & \leftarrow \text{get} \\
  \text{if } s & > 0 \\
  \text{then put } (s-1) & >> \text{loop} \\
  \text{else return } s
\end{align*}
\]
How can we speed things up?
Outline

- One handler rule
- Linearity
- Handler resolution
One handler rule

For each delimiter there is a unique handler
One handler rule

For each delimiter there is a unique handler

Contrast with shift/reset
One handler rule

For each delimiter there is a unique handler

Contrast with shift/reset

Optimize multiple calls
Linearity

\[
\begin{align*}
\text{state} &:: s \to [\text{State } s] a \to (a, s) \\
\text{state } s \ [\text{get} \ ? \ k] & = \text{state } s \ ? \ k \ s \\
\text{state } _{-} \ [\text{put} \ s \ ? \ k] & = \text{state } s \ ? \ k \ () \\
\text{state } s \ [a] & = (a, s)
\end{align*}
\]
Linearity

(Exactly) one invocation

\[
\text{state} :: s \to \text{[State } s\text{]} \to a \to (a,s)
\]

\[
\text{state } s \text{ [get } ? \text{ k]} = \text{state } s \text{ ? k s}
\]

\[
\text{state } _{-} \text{ [put } s \text{ ? k]} = \text{state } s \text{ ? k ()}
\]

\[
\text{state } s \text{ [a]} = (a,s)
\]
Linearity

(Exactly) one invocation

```
state :: s -> [State s] a -> (a,s)
state s [get ? k] = state s
state _ [put s ? k] = state s
state s [a] = (a,s)
```

Tail call
Linearity

Stacks before / after operations are almost identical.

Avoid copying the stack!
Linearity

- Analyze linearity of handlers.
- Run the handler code on top of the current stack.
- Handle ‘resources’ of handlers explicitly outside of the control stack.
Handler resolution

- Dynamic search is impractical.
- Static effect information available.
- Vector of stack pointers?
Conclusion
Summary

Great way to model effects

Too slow for taking over the world tonight

Working on a solution for tomorrow night
Questions

Type systems

Program analyses

Implementation techniques